INRIA

UNITE DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 961

Programme 2

RÉSOLUTION DU PROBLÈME DE MULTIKNAPSACK EN PARALLÈLE : algorithme PR²88

**PLATEAU Gérard, ROUCAIROL Catherine,
GACHET Sylvie**

Janvier 1989

RESOLUTION DU PROBLEME DE MULTIKNAPSACK EN PARALLELE : algorithme PR²88

PARALLEL RESOLUTION OF THE MULTIKNAPSACK PROLEM : algorithme PR²88

Plateau Gérard [*], Roucairol Catherine [**], Gachet Sylvie [***]

[*] Université Paris-Nord

[**] Université Paris VI-Masi et INRIA

[**] Ingénieur IIE, stagiaire INRIA 88

RESOLUTION DU PROBLEME DE MULTIKNAPSACK EN PARALLELE: algorithme PR^{2}88

Plateau Gérard*, Roucairol Catherine **, Gachet Sylvie***

Résumé:
Les facilités offertes par les machines parallèles (vectorisation, multitâches) sont exploitées pour résoudre des problèmes de multiknapsack en variables 0-1:
- dans une première phase, plusieurs tests lancés en parallèle permettent de réduire la taille du problème(par fixation de variables et élimination de contraintes),
- dans la seconde, la solution optimale du problème est trouvée grâce à une méthode de recherche arborescente (Branch & Bound) elle aussi menée en parallèle.
L'algorithme parallèle correspondant a été implémenté sur une machine multiprocesseur à mémoire partagée: le CRAY2. Les résultats obtenus sur des exemples célèbres de la littérature sont reportés.

Mots-clés:
optimisation pseudo-booléenne,réduction,relaxation composite,algorithme Branch &Bound parallèle,machine multiprocesseurs asynchrone.

Abstract:
The characteristics of parallel machines (vectorization, multiprocessing) are exploited in order to solve the 0-1 multiknapsack problem:
-in a first phase, a lost of tests are performed in parallel in order to reduce the size of the problem (fixation of variables, elimination of constraints)
-in a second phase, a parallel Branch and Bound algorithm allows to get an optimal solution.
Our parallel algorithm has been implemented on the asynchronous multiprocessor machine CRAY 2. Computational results are reported and compared with those obtained in a sequential approach.

Keywords:
pseudo boolean optimization, reduction, surrogate relaxation, parallel Branch & Bound algorithms, asynchronous multiprocessor machine.

PLAN

SECTION 1

A SUPERCOMPUTER ALGORITHM FOR THE 0-1 MULTIKNAPSACK PROBLEM

Contents

SECTION 2

UN ALGORITHME PARALLELE POUR LA RESOLUTION DU MULTIKNAPSACK:

PR^2 88

Sommaire:

SECTION 1

A SUPERCOMPUTER ALGORITHM FOR THE 0-1 MULTIKNAPSACK PROBLEM

A SUPERCOMPUTER ALGORITHM FOR THE 0-1 MULTIKNAPSACK PROBLEM

Gérard PLATEAU
University Paris-Nord - CSP
Département de Mathématiques
et Informatique
Avenue Jean-Baptiste Clément
93430 Villetaneuse, France

and

Catherine ROUCAIROL
University Paris 6, and,
INRIA - Rocquencourt
B.P. 105
78153 Le Chesnay Cédex, France

ABSTRACT

The characteristics of parallel machines (vectorization, multiprocessing) are exploited in order to solve the 0-1 multiknapsack problem :

- in a first phase, a lot of tests are performed in parallel in order to reduce the size of the problem (fixation of variables, elimination of constraints)
- in a second phase, a parallel branch and bound algorithm allows to get an optimal solution.

Our parallel algorithm has been implemented on the asynchronous multiprocessor machine CRAY 2. Computational results are reported and compared with those obtained in a sequential approach.

Keywords : *Pseudo boolean optimization, reduction, surrogate relaxation, parallel Branch and Bound algorithms, asynchronous multiprocessors machine.*

1. INTRODUCTION

Although NP-Hard, the 0-1 knapsack problem is well solved in a sequential way for many classes of instances. The lot of exact algorithms in literature includes implicit enumeration procedures with an experimental linear time complexity (see for example [Balas and Zemel 1980 ; Fayard and Plateau 1982 ; Martello and Toth 1978 and 1988 ; Plateau and Elkihel 1985]). Sizes up to several thousand variables are easily reached for some instances whose data are randomly generated from a uniform distribution.

On the opposite, optimal solutions of the 0-1 multidimensional knapsack problem are classically reached for instances with strongly limited sizes : about ten constraints and one

hundred variables [Fleisher 1976 ; Freville and Plateau 1986 and 1987 ; Gavish and Pirkul 1985 ; Petersen 1967 ; Plateau 1976 ; Senju and Toyoda 1968 ; Shih 1979 ; Weingartner and Ness 1967]

Up to now, parallel algorithms devoted to the knapsack problem have only been designed for the one dimension case [Gopalakshnan et al. 1987 ; Karnin 1984 ; Klein et al. 1983 ; Yao 1982]. In addition, all these works are based upon the use of theoretical models of parallel computation (PRAM with a polynomial - even exponential in the input size - number of processors).

The aim of our work is to elaborate an efficient parallel algorithm for solving the 0-1 multiknapsack problem and to implement it on an actual supercomputer whose number of processors is obviously independent of the input sizes.

The solving of the 0-1 multiknapsack problems includes two main phases :

- the reduction of the size : computations of lower bounds (heuristic methods) and upper bounds (Lagrangean and surrogate relaxations) allow to fix variables at there optimal values and to drop redundant constraints,
- the implicit enumeration of the reduced problem.

The basic idea of this paper is to exploit first the speed of supercomputers in order to perform much more works than in a sequential way and thus to improve the size reduction (that is to increase the number of fixed variables and eliminated constraints).

In addition, a parallel branch and bound algorithm including this reduction scheme is designed by exploiting previous studies for various combinatorial optimization problems [Lavallée and Roucairol 1985 ; Roucairol 1987a and 1987b]. It allows to speed up the implicit enumeration of the tree search nodes.

2. PROBLEM STATEMENT

The following 0-1 multiconstraint knapsack problem (B) is considered :

$$\begin{array}{ll} \text{maximize} & cx \\ \text{subject to} & Ax \leq b \; ; \; x \in V \; , \end{array}$$

whose data are such that

$c \in \mathbb{N}^{n}_{*}$, $b \in \mathbb{N}^{m}_{*}$, A is a mxn dense non-negative integer matrix,
and where $V = \{x \in \mathbb{R}^{n} \mid x_j = 0 \text{ or } 1, \; j=1,2, \ldots , n\}$.

3. NOTATIONS

Given an optimization problem (P) :

$v(P)$: optimal value of (P) ;

$\bar{v}(P)$ (resp. $\underline{v}(P)$) : upper (resp. lower) bound on $v(P)$;

$(P \mid x \in X)$: (P) with the added constraint $x \in X$.

4. REDUCTION TESTS

The use of a preprocessing reduction algorithm improves the efficiency of exact methods by decreasing the size of the problems (fixation of variables and elimination of constraints) (see [Freville and Plateau 1986 and 1987]).

The classical concept of surrogate constraints introduced by Glover [1965] allows to generate reduction tools with expected linear time complexities (use for the more sophisticated tests of the solving of knapsack relaxations (see [Fayard and Plateau 1977 and 1982, Freville and Plateau 1986]).

Before to give the statement of these duality based reduction tests, the simple following ones allow the construction of a so-called *well-stated* problem.

4.1. The well-stated problem

This problem is obtained by dropping obviously redundant constraints and fixed variables

(R_1) if there exists an index $j \in \{1,2,...,n\}$ and an index $i \in \{1,2,...,m\}$ such that :
$a_{ij} > b_i$
then the variable x_j must be fixed at 0.

(R_2) if there exists an index $i \in \{1,2,...,m\}$ such that :

$$\sum_{j=1}^{n} a_{ij} \leq b_i$$

then the constraint i must be eliminated.

(R_3) if there exists an index $j \in \{1,2,...,n\}$ such that :

$A^j = 0$

then the variable x_j must be fixed at 1.

4.2. Duality based tests

For a well-stated problem (B) with n variables and m constraints, given a multiplier $w \in \mathbb{R}_+^m$ (generated or not by the solving of the Lagrangean (or surrogate) dual of (B)), let us consider :

- the associated surrogate relaxation (0-1 knapsack problem derived from (B)) :

$$(B^0(w)) \qquad \max \; cx \qquad \text{s.t.} \; wAx \leq wb \; ; \; x \in V$$

- for each q in $\{1,2,...,m\}$, the following 0-1 knapsack problem

$$(B^q(w)) \qquad \max \; A_q x \qquad \text{s.t.} \; wAx \leq wb \; ; \; x \in V$$

4.2.1. *Fixation of variables*

With the aim of finding a solution with a value greater than a given lower bound $\underline{v}(B)$ on v(B), the fixation of variables at their optimal values is realized as follows :

Theorem 1.

If there exists an index $j \in \{1,2,...,n\}$ and $\epsilon \in \{0,1\}$ such that

$$\bar{v}(B^0(w) \mid x_j = \epsilon) \leq \underline{v}(B)$$

then the variable x_j must be fixed at the value $1-\epsilon$.

Proof : see [Freville and Plateau 1986 ; Plateau 1976].

4.2.2. *Elimination of constraints*

The following result gives a sufficient condition for dropping redundant constraints :

Theorem 2.

If there exists an index q in {1,2,...,m} such that

$$\bar{v}(B^q(w)) \leq b_q$$

then the constraint $A_q x \leq b_q$ can be eliminated.

Proof : see [Freville and Plateau 1986, Plateau 1976].

4.3. Main features of the serial reduction algorithm

The serial reduction system FPR83 realized by Fréville and Plateau includes three main phases :

(i) computation of an upper bound on the optimal value by using Lagrangean and surrogate relaxations.

(ii) computation of a lower bound by using the so-called AGNES heuristic methods.

(iii) size reduction by the conjunction of tests applied to 0-1 knapsack problems derived from (B) as described in theorems 1 and 2.

This serial algorithm FPR83 is extensively described in [Fréville and Plateau 1986 and 1987] with the actual chain of reduction tests and additional details about the selected options for its implementation.

It has been tested on a CII HB IRIS 80 with twenty concrete problems [Fleisher 1976 ; Petersen 1967 ; Plateau 1976 ; Senju and Toyoda 1968 ; Weingartner and Ness 1967] and thirty randomly generated problems [Shih 1979]. These computational results prove the quality of performances of heuristics AGNES and the decreasing of the running times for exact methods including this prepocessing reduction system FPR83.

For example, for the seven problems due to Petersen [1967], table 1 details for each of them their original and reduced sizes, and the running times (in second) for their exact solving by Shih's code [1979] respectively without (original size) and with (reduced size) the inclusion of FPR83.

original size	time (second)	reduced size	time reduction	global
10 x 6	.2	0 x 0	1.2	1.2
10 x 10	.5	1 x 3	2.3	2.3
10 x 15	1.9	5 x 8	4.8	4.8
10 x 20	1.8	2 x 8	5.7	5.8
10 x 28	9.2	2 x 9	5.4	5.5
5 x 39	>25	4 x 28	7.5	25
5 x 50	>25	4 x 38	7.7	28.8

table 1

5. HIGH-LEVEL DESCRIPTION OF THE PARALLEL ALGORITHM

5.1. Reduction phase

Instead of constructing the optimal dual multiplier w^* of (B), the aim of our parallel algorithm is to generate two sets W_1 and W_2 of dual multipliers which allow to produce a set of 0-1 knapsack problems :

. fixation of variables :

$$(B^0(w))\ \max cx \qquad \text{s.t.} \quad wAx \leq wb\ ;\ x \in V\ ,\ w \in W_1$$

. elimination of constraints : $q \in \{1,\ldots,m\}$

$$(B^q(w))\ \max A_q x \quad \text{s.t.} \quad wAx \leq wb\ ;\ x \in V\ ,\ w \in W_2\ .$$

This idea is motivated by the two following observations.

First, it is not proved that the use of the unique 0-1 knapsack problem $(B^0(w^*))$ leads to the best results as concerns the number of fixed variables and eliminated constraints. In addition, the relation

$$\bar{v}(B^0(w)) \leq \bar{v}(B^0(w')) \qquad w,\ w' \in \mathbb{R}^m_+$$

does not imply necessarily for any variable x_j and $\epsilon \in \{0,1\}$

$$\bar{v}(B^0(w) \mid x_j = \epsilon) \leq \bar{v}(B^0(w') \mid x_j = \epsilon).$$

Second, from a computational time point of view, it is not realistic to consider serially sets of tool knapsacks $(B^0(w))$, $w \in W_1$ and $w \in W_2$. However, this kind of approach had been used by Hansen and Plateau [1976] in order to test experimentally its robustness.

Obviously, due to parallelism, it is possible to work concurrently with several tool knapsacks and to apply a sequence of reduction tests for each of them.

The distributed work is the following :

- fixation of variables :

  given a multiplier $w \in W_1$, perform the usual sequence of tests on $(B^0(w))$.

- elimination of constraints :

  given a constraint $q \in \{1,2,...,m\}$ candidate for elimination, use of another sequence of tool knapsacks $(B^q(w))$, $w \in W_2$.

In fact, the 0-1 knapsack problems $(B^0(w)) \mid x_j = \epsilon)$ and $(B^q(w))$ are not exactly solved, but with respect to theorems 1 and 2 these different kinds of relaxations are used :

. Lagrangean relaxations
. the associated linear programs.

This parallel reduction algorithm is an extended version of a previous algorithm denoted by $PR^2 87$ [Plateau and Roucairol 1987].

5.1.1. *Fixation of variables*

Namely, the following results are applied : given a tool knapsack $(B^0(w))$, $w \in W_1$, let us define its Lagrangean relaxation associated with a multiplier $\lambda \in \mathbb{R}^+$:

$LR(\lambda)$ ma $cx + \lambda (wb - wAx)$ s.t. $x \in V$

Theorem 3.

(i) the function $\lambda \rightarrow v(LR(\lambda))$ is a convex piece-wise linear function.
(ii) the breakpoints of this function are $\lambda_j = c_j / wA^j$, $j=1,2,...,n$.

Due to theorems 1 and 3, this result is deduced :

Corollary :

For the fixation of any variable of (B) by Lagrangean relaxation, it is sufficient to consider the finite set $\{LR(\lambda)\}$, $\lambda \in \Lambda = \{0,\lambda_1,...,\lambda_n,+\infty\}$.

Proof : see [Hammer, Padberg and Peled 1975].

One more time, instead of constructing the optimal Lagrangean dual multiplier λ^* of $(B^O(w))$, we generate the set Λ of multipliers ; this leads to an obvious vectorizable reduction structure.

Let us recall that for each λ in Λ, by denoting :

$$J^+(\lambda) = \{j\in\{1,...,n\} \mid c_j - \lambda w A^j > 0\} \; ; \; J^-(\lambda) = \{j\in\{1,...,n\} \mid c_j - \lambda w A^j < 0\}$$

then

$$v(LR(\lambda)) = \lambda w b + \sum_{j\in J^+(\lambda)} (c_j - \lambda w A^j)$$

and for j in $J^+(\lambda)$:

$$v(LR(\lambda) \mid x_j=1) = v(LR(\lambda)) \; ; \; v(LR(\lambda) \mid x_j=0) = v(LR(\lambda)) - (c_j - \lambda w A^j)$$

and for j in $J^-(\lambda)$:

$$v(LR(\lambda) \mid x_j=0) = v(LR(\lambda)) \; ; \; v(LR(\lambda) \mid x_j=1) = v(LR(\lambda)) + (c_j - \lambda w A^j)$$

Thus, for each variable x_j, $j\in J^+(\lambda) \cup J^-(\lambda)$, it is sufficient to exploit the comparison of these two vectors :

$$(|c_j - \lambda w A^j|)_{\lambda\in\Lambda} \quad \text{and} \quad (v(LR(\lambda)) - \underline{v}(B))_{\lambda\in\Lambda}$$

Moreover, it is important to point out the following result :

Theorem 4.

The set of fixed variables by using all the Lagrangean multipliers includes the set that could be obtained by solving the linear programs associated with the knapsack problems

$$(B^O(w) \mid x_j = \epsilon) \; , \; \epsilon \in \{0,1\} \; .$$

5.1.2. *Binary relations*

The previous results can be improved by using binary relations in the following way (Fayard and Plateau 1977, Jaumard 1986]. Given

$$\bar{\lambda} \in \Lambda, \; k \in \{1,...,n\} \text{ and } \epsilon = \begin{cases} 0 & \text{if } k \in J^+(\bar{\lambda}) \\ 1 & \text{if } k \in J^-(\bar{\lambda}) \end{cases}$$

we first search all the fixed variables by using the other Lagrangean multipliers when x_k is fixed at ϵ.

It means that these sets are constructed :

$$X_1 = \{j \mid \exists \lambda \in \bar{\Lambda}\ \bar{\lambda} : v(LR(\lambda) \mid x_k = \epsilon\ ;\ x_j=0) \leq \underline{v}(B)\}$$

and

$$X_0 = \{j \mid \exists \lambda \in \bar{\Lambda}\ \bar{\lambda} : v(LR(\lambda) \mid x_k = \epsilon\ ;\ x_j=1) \leq \underline{v}(B)\}\ .$$

Then we can strenghten the previous reduction results by comparing these two bounds :

$$v(LR(\bar{\lambda}) \mid x_k=\epsilon\ ;\ x_j=1 \quad \forall\, j \in X_1\ ;\ x_j=0 \quad \forall\, j \in X_0)$$

and $\underline{v}(B)$.

5.1.3. *Elimination of constraints*

Finally, as concerns the elimination of constraints, we solve linear programs associated with $(B^q(w))$, $w \in W_2$ by applying the expected linear time complexity algorithm described in [Fayard and Plateau 1977 and 1982].

5.2. Branch and Bound phase

Our parallel Branch and Bound algorithm includes the reduction scheme and exploits previous studies for various combinatorial optimization problems. [Lavallée and Roucairol 1985 ; Roucairol 1987 a and b].

It is an asynchronous parallel algorithm based upon the case of a bounded number of processes which are to be dealt with concurrently.

The distribution of the work among the different processes is done by giving access to a shared list which contains information about every node which is to be expanded. This list is implemented as a heap in order to use fast algorithms to insert, sort and remove items. As a best first strategy is used, every active process finds at the top of the list (node with the best upper bound) the node it is going to expand (i.e. to branch following the value of the basic variable of the continuous surrogate knapsack associated with this node and to evaluate the created successors).

In the shared list, insertion of items is done whenever the expansion of a node generates several successors whose evaluation is greater than the best known lower bound (feasible solution) ; these successors are then inserted in the list.

The best known lower bound (BKLB) is a shared variable which is updated whenever a local lower bound (llb), greater than BKLB, is found at a node to be expanded. Items are suppressed either at the beginning or at the end of the list. The former case occurs whenever

an inactive process looks for a new job, the latter case occurs whenever a llb is greater than BKLB : every node whose evaluation is less than llb is eliminated because it cannot lead to an optimal solution.

The algorithm terminates whenever the list is empty and all the processes are inactive.

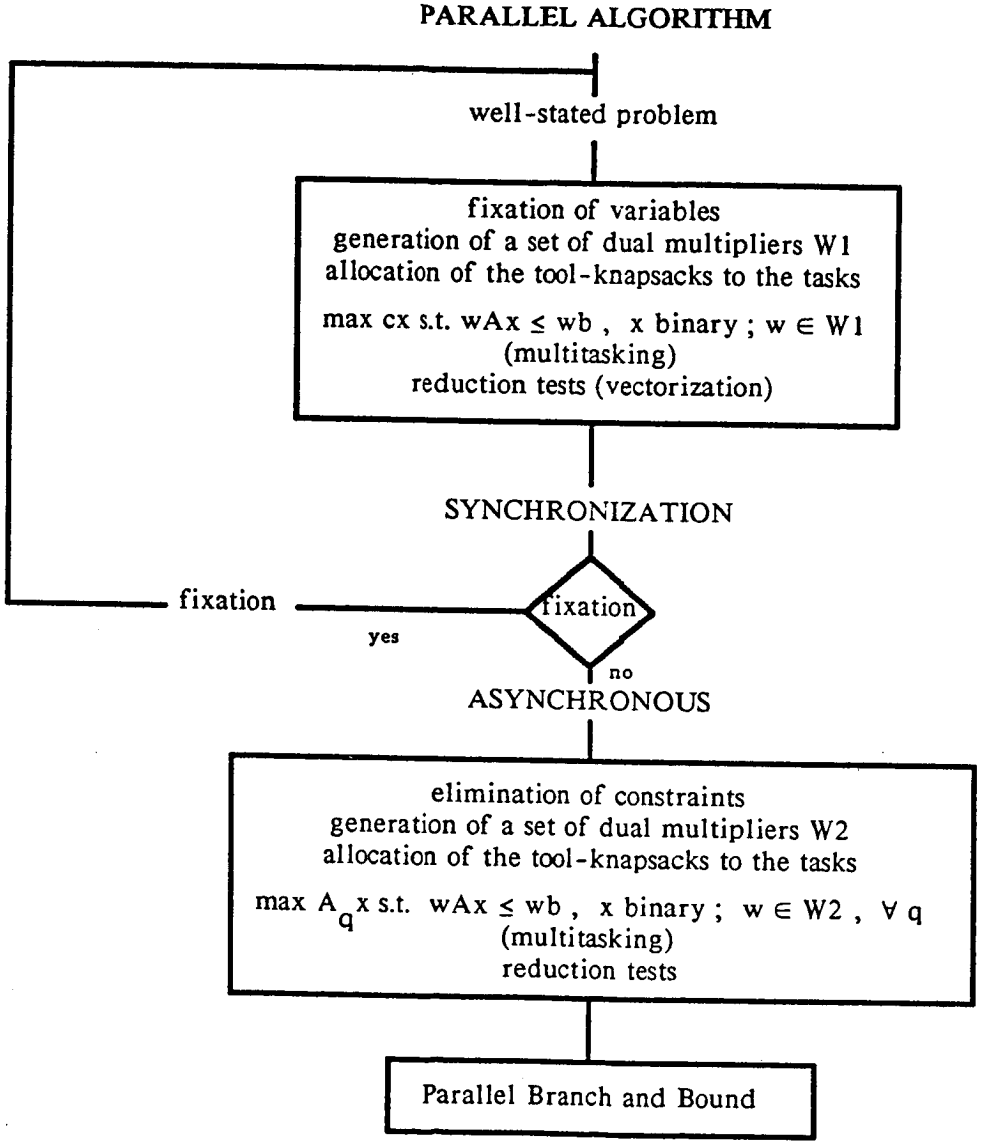


Figure 1.

6. EXPERIMENTAL RESULTS

6.1. Implementation

Our algorithm has been implemented on the asynchronous multiprocessor CRAY2. Its four vector processors communicate by simultaneous reading or exclusive writing on a shared central memory (32Mega words, CPU cycle time 4 ns for vector operations).

The code has been written in Fortran 77 [Plateau, Roucairol and Gachet 1988] and the multitasking library of CRAY has been used for synchronization and communication operations.

6.2. Numerical results

Computational results are summarized by considering a lot of test problems including the fifty problems from literature reported in table 2, in order to realize comparisons with the sequential algorithms.

Author	Number of problems	Size from	Size to
Shih	30	5 x 30	5 x 90
Ness, Weingartner	8	2 x 28	2 x 105
Petersen	7	10 x 10	10 x 28
		5 x 39	5 x 52
Senju, Toyoda	2	30 x 60	30 x 90
Hansen, Plateau	2	4 x 28	4 x 35
Fleisher	1	10 x 20	10 x 20

table 2

Tables 3 and 4 contain comparisons of the number of variables fixed by our parallel algorithm denoted by PR² 88 and the sequential algorithm FPR 83.

Problem	Size	Size after reduction FPR 83	Size after reduction PR² 88
Petersen	10x6	0x0	0x0
	10x10	1x3	1x3
	10x15	5x8	5x8
	10x20	2x8	2x7
	10x28	2x9	2x6
	5x39	4x28	4x27
	5x50	4x38	4x36
Ness Weingartner	2x28	1x4	1x4
	2x28	1x5	0x0
	2x28	2x19	2x14
	2x28	1x5	1x5
	2x28	0x0	0x0
	2x28	2x7	2x5
	2x105	2x12	2x12
	2x105	2x29	2x28
Hansen Plateau	4x28	4x27	4x26
	4x35	4x34	4x34
Senju Toyoda	30x60	30x40	30x40
	30x60	30x37	26x33

table 3

Our algorithm is always better or at least as good as the sequential one.

Problem	Size	Size after reduction	
		FPR 83	PR² 88
Shih	5x30	5x14	5x12
	5x30	1x5	0x0
	5x30	4x11	4x7
	5x30	1x2	0x0
	5x30	0x0	0x0
	5x40	3x11	2x10
	5x40	3x11	3x8
	5x40	2x10	2x5
	5x40	1x3	1x3
	5x50	3x18	3x7
	5x50	2x12	2x12
	5x50	1x5	1x5
	5x50	3x15	0x0
	5x60	3x11	3x11
	5x60	3x6	3x6
	5x60	2x8	2x8
	5x60	1x14	0x0
	5x70	3x13	3x11
	5x70	1x2	1x2
	5x70	3x9	3x9
	5x70	1x7	1x5
	5x80	3x29	3x10
	5x80	2x13	2x13
	5x80	3x20	3x20
	5x80	3x10	3x9
	5x90	3x30	0x0
	5x90	2x7	2x7
	5x90	2x8	0x0
	5x90	2x9	0x0
	5x90	0x0	

table 4

We are now conducting experiments for the global algorithm, several previous problems have been already exactly solved in very short computer time. The time ratio is about one hundred as compared with FPR 83 ran on a CII HB IBIS 80 computer.

We report below the results obtained on the most difficult problem from the litterature [Fleisher 1976].

The table 5 shows some problems of granularity due to the small sizes of litterature's instances.

In order to balance the load of each processor, we decide to define a priority rule for accessing the shared list of node. Namely, a process is allowed to treat a node if its current load does not exceed t% of the mean load. The running times are in seconds. For the best result, a gain of 41% is obtained for t equals 50.

		Processors			
	t %	1	2	3	4
N p	0	79 25%	123 34%	89 22%	67 19%
N p q	10%	92 26% 2	86 24% 2	91 25% 24	89 25% 3
N p q	30%	84 23% 0	95 27% 2	87 24% 0	92 26% 10
N p q	50%	95 26,5% 0	95 26,5% 3	80 22% 0	88 25% 0

Table 5 - Granularity's problem on Fleisher's example
(N number of nodes treated by a processor, p percentage of work done by a processor, q number of times a processor waits)

7. CONCLUSION

The results of our first experiments show the various interests of the use of parallel computing for combinatorial optimization :

- to perform much more works than in a sequential way in order to get more informations and to improve the size reduction,

- to accelerate the search of an optimal solution in a Branch and Bound tree.

It is clear that multiprocessors machine will be much more attractive for instances with a number of variables and constraints greater than those of litterature. This will be the future direction of our work.

REFERENCES

Balas E. and Zemel E. (1980), "An algorithm for Large Zero-One Knapsack Problem", *Operations Research*, 28, 1130-1154.

Fayard D. and Plateau G. (1982), "An algorithm for the Solution of the 0-1 Knapsack Problem", *Computing* 28, 269-287.

Fayard D. and Plateau G. (1977), "Reduction algorithms for single and multiple constraints 0-1 linear programming problems", *Proceeding of the Congress Methods of Mathematical Programming*, Zakopane, Poland.

Fleisher J. (1976), *Sigmap Newsletter* 20

Freville A. and Plateau G. (1986), "Heuristics and reduction methods for multiple constraints 0-1 linear programming problems", *EJOR* 24, 206-215.

Freville A. and Plateau G. (1987), "Hard 0-1 multiknapsack test problems for size reduction methods", *Research Report* 72, Université Paris-Nord.

Gavish B. and Pirkul H. (1985), "Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality, *Mathematical Programming* 31, 78-105.

Glover F. (1965), "A multiphase dual algorithm for the 0-1 integer programming problem, *Operations Research* 13, 879-919.

Gopalakrishnan P.S., Kanal L.N. and Ramakrishnan I.V. (1987), "Approximate algorithms for the Knapsack problem on Parallel computers, *IBM Report* RC 12549, 1-34.

Hammer P.L., Padberg M.W. and Peled U.N. (1975), "Constraint pairing in integer programming", *INFOR - Canadian Journal of Operational research and Information Processing* 13, 68-81.

Jaumard B. (1986), "Extraction et utilisation de relations booléennes pour la résolution des programmes linéaires en variables 0-1", *Thèse de Doctorat*, ENSET.

Karnin E.D. (1984), "A parallel algorithm for the knapsack problem, *IEEE Trans. on Computers* C-33, 404-408.

Klein P. and Meyer auf der Heide F. (1983), "A lower time bound for knapsack problem on random access machine", *Acta Informatica* 19, 385-395.

Lavallée I. and Roucairol C. (1985), "Parallel branch and bound algorithms", *EURO VII Congress, Research Report MASI* 112, Université Paris 6.

Martello S. and Toth P. (1978), "Algorithm for the solution of the 0-1 single knapsack", *Computing* 21, 81-86.

Martello S. and Toth P. (1988), "A new algorithm for the 0-1 knapsack problem", Management Science 34, 633-644.

Petersen C.C. (1967), "Computational experience with variants of the Balas algorithm applied to the selection of R and D projects", *Management Science* 13 (9), 736-750.

Plateau G. (1976), "Réduction de la taille des problèmes linéaires en variables 0-1", *Research Report* 71, UST Lille 1.

Plateau G. and Elkihel M. (1985), "A hybrid method for the 0-1 knapsack problem", *Methods of Operations research* 49, 277-293.

Plateau G. and Roucairol C. (1987), "Algorithm PR²87 for the parallel size reduction of the 0-1 multiknapsack problem" *First Colloquium on Boolean methods for combinatorial optimization*, Chexbres.

Plateau G., Roucairol C. and Gachet S. (1988), "Algorithm PR²88 for the parallel resolution of the 0-1 multiknapsack problem", *Research Report INRIA* to appear.

Roucairol C. (1987a), "A parallel branch and bound algorithm for the quadratic assignment problem", *Discrete Applied Mathematics* 18, 211-225.

Roucairol C. (1987b), "Du sequentiel au parallèle : la recherche arborescente et son application à la programmation quadratique en variables 0-1, *Thèse d'Etat*, Université Paris 6.

Roucairol C. (1988), "Parallel Computing in Combinatorial Optimization", *Proceedings of Numerical methods for parallel vector computers*, North Holland, to appear 1989.

Senju S. and Toyoda Y. (1968), "An approach to linear programming with 0-1 variables", *Management Science* 15 (4), 196-207.

Shih W. (1979), "A branch and bound method for the multiconstraint 0-1 knapsack problem", *Journal of the Operational Research Society* 30 (4), 369-378.

Weingartner H.M. and Ness D.N. (1967), "Methods for the solution of the multidimensional 0-1 knapsack problem", *Operations Research* 15 (1), 83-103.

Yao A.C. (1982), "On parallel computation for the knapsack problem", *JACM* 29 (3), 898-903.

SECTION 2

UN ALGORITHME PARALLELE POUR LA RESOLUTION DU MULTIKNAPSACK: PR^2 88

CHAPITRE 1 : INTRODUCTION

I- PRESENTATION DU PROBLEME DU MULTIKNAPSACK

Le problème consiste en la maximisation d'une fonction bivalente à coefficients entiers sous plusieurs contraintes linéaires à coefficients entiers .

Soit le problème (B) à n variables et m contraintes . Nous pouvons formaliser ce problème de la manière suivante :

$$\begin{aligned} (B) \quad & \text{Max } c.x \\ & \text{s.c. } A.x \leq b \\ & x \in \{0,1\}^n \end{aligned}$$

A est la matrice des contraintes , de dimension mxn , à coefficients entiers .

b est la matrice des seconds membres des contraintes , de dimension mx1 , à coefficients entiers .

c est la matrice de la fonction économique , de dimension 1xn , à coefficients entiers .

Nous nous proposons donc d'étudier la réduction de la taille de ce problème et sa résolution .

Nous dirons qu'un point x de $\{0,1\}^n$ est une **solution réalisable** de (B) s'il vérifie les contraintes de (B) , c'est-à-dire si $A.x \leq b$.

Nous dirons qu'une solution réalisable x^* de (B) est une **solution optimale** de (B) si elle maximise la fonction économique de (B) , c'est-à-dire si :

$$c.x^* = \max \{ c.x \mid x \text{ est une solution réalisable de (B)} \} .$$

II- PRINCIPE DE LA RESOLUTION D'UN MULTIKNAPSACK

La résolution du problème de multiknapsack que nous proposons se déroule en trois phases:

- Détermination d'un minorant de la valeur du problème
- Réduction de la taille du problème
- Résolution du problème réduit par une méthode d'énumération implicite

La réduction consiste en l'élimination de variables et de contraintes . Eliminer des variables signifie les fixer à 0 ou à 1 . Eliminer des contraintes signifie montrer qu'elles sont redondantes dans le système de contraintes du problème source et les supprimer .

Le but de la réduction est d'obtenir un problème de taille réduite , donc plus facile à résoudre par quelque algorithme que ce soit . En effet , il est montré (Fréville et Plateau [FR1]) , que pour des problèmes de tailles suffisamment grandes , on a :

(temps de réduction) (+ temps de résolution) (du problème réduit) $\leq$ (temps de résolution) (du problème original)

III- NOTATIONS

Les notations utilisées seront les suivantes :

Etant donné un ensemble J :

[J] : enveloppe convexe de J

Etant donnée une matrice de format (I*J) où I et J sont des ensembles finis :

a_{ij} : elément (i,j) de A

A^j : colonne j de A

A_i : ligne i de A

Etant donné un vecteur (ligne ou colonne) indicé par un ensemble fini J:

x_j : elément d'indice j de x

Etant donné un problème d'optimisation (B) à n variables bivalentes :

v(B) : valeur optimale de (B) (= max {c.x | x solution réalisable})

$\underline{v}(B)$: minorant de v(B)

$\overline{v}(B)$: majorant de v(B)

$\underline{x}$: une solution réalisable de (B)

F(B) : l'ensemble des solution réalisables de (B)

x^* : une solution optimale de (B)

$(B|x_j = \varepsilon)$: lorsque (B) est résolu avec la condition supplementaire $x_j=\varepsilon$ $(\varepsilon \in \{0,1\})$

$x_j <- \varepsilon$: x_j est fixée à ε $(\varepsilon \in \{0,1\})$

V : = $\{ x \in \{0,1\}^n \}$

$(\overline{B})$: problème (B) dans lequel x parcourt $[0,1]^n$

Soient :
- $w \in R_+^m$, un multiplicateur composite
- i une contrainte
- t une contrainte

Considérons les quatre types de knapsacks suivants :

$$(B^\circ(w)) \quad \max cx \quad \text{s.c. } wAx \leq wb \quad x \in \{0,1\}^n$$

$$(B^\circ_i) \quad \max cx \quad \text{s.c. } A_i x \leq b_i \quad x \in \{0,1\}^n \quad (w = (0,\ldots,\overset{\overset{i}{\downarrow}}{1},\ldots,0))$$

$$(B^t(w)) \quad \max A_t x \quad \text{s.c. } wAx \leq wb \quad x \in \{0,1\}^n$$

$$(B^t_i) \quad \max A_t x \quad \text{s.c. } A_i x \leq b_i \quad x \in \{0,1\}^n$$

CHAPITRE 2 : REDUCTION DE LA TAILLE D'UN MULTIKNAPSACK

Nous distinguons deux types de réduction : les réductions triviales qui ramènent le problème à un problème bien posé et les réductions issues de tests plus élaborés . Nous allons exposer les principes des réductions triviales , de la fixation des variables et de l'élimination des contraintes .

I- LES REDUCTIONS TRIVIALES

Certaines contraintes et certaines variables peuvent être éliminées de manière triviale . Nous allons exposer le principe de ces éliminations . ([FA1],[FR1])

1- Test R1

Ce test permet de fixer à 0 les variables qui ont une contribution trop élevée dans une contrainte .

Soient une contrainte i et une variable x_j .

Si $a_{ij} > b_i$ alors la variable x_j peut être fixée à 0.

2- Test R2

Nous éliminons les "fausses contraintes" , c'est-à-dire les contraintes i qui sont vérifiées par tous les éléments de V

$$\forall x \in V \quad \Sigma_j a_{ij} * x_j \leq b_i$$

Nous pouvons montrer qu'une contrainte est une "fausse contrainte" si et seulement si elle vérifie la condition suivante :

$$\Sigma_j a_{ij} \leq b_i$$

Conclusion
Soit i une contrainte
Si $\Sigma_j a_{ij} \leq b_i$ alors la contrainte i peut être éliminée .

3- Test R3

Suite aux réductions précédentes , certaines variables peuvent être absentes de toutes les contraintes non éliminées . Nous fixons les variables qui n'apparaissent dans aucune contrainte . De plus , pour maximiser le coût nous les fixons à 1 .

Si $\forall i \in \langle 1,m \rangle$ $a_{ij} = 0$ alors la variable x_j est fixée à 1
(la colonne i de la matrice A est nulle)

II- FIXATION DES VARIABLES
([FA1],[FR1])

Nous nous proposons de fixer certaines variables du problème à 1 ou 0 . Nous allons exposer le principe de cette fixation qui est fondée sur la recherche d'une solution meilleure que la meilleure solution connue $\underline{x}$ (à laquelle correspond un coût $\underline{v}(B)$) .

Soit x_j une variable du problème

Soit $\varepsilon \in \{0,1\}$

Soit $F(B|x_j=\varepsilon)$ le sous-ensemble de F(B) contenant les solutions réalisables de (B) pour lesquelles x_j vaut ε :

$$F(B|x_j=\varepsilon) = \{x \in (0,1) \mid x_j = \varepsilon \text{ et } Ax \leq b\}$$
$$= \{x \in F(B) \mid x_j = \varepsilon\}$$

Soit $\underline{v}(B)$ un minorant de la valeur de (B)

Si les valeurs de toutes les solutions éléments de $F(B|x_j=\varepsilon)$ sont strictement inférieures à $\underline{v}(B)$ alors la variables x_j peut être fixée à $1-\varepsilon$. Cela signifie que toutes les solutions dont la j^{iem} composante est fixée à ε sont moins bonnes que la solution $\underline{x}$. Il faut donc donner la valeur $1-\varepsilon$ à x_j .

$(\forall x \in F(B\|x_j=\varepsilon)\ \ c.x \leq \underline{v}(P)) \Rightarrow x_j$ **doit être fixée à** $1-\varepsilon$

Nous nous proposons de calculer un majorant des valeurs des eléments de $F(B|x_j=\varepsilon)$.

Considérons le problème B_j^{ε} (ou $B|x_j=\varepsilon$) la restriction du problème P lorsque x_j vaut ε . Soit :

$$(B_j^{\varepsilon}) \quad \max \Sigma_{i \neq j}\ c_i.x_i + \varepsilon c_j$$

$$\text{s.c. } \Sigma_{k \neq j}\ a_{i,k}x_k \leq b_k - \varepsilon a_{ij} \quad (i = 1,m)$$
$$x \in \{0,1\}^n$$

Nous remarquons que $v(B_j^{\varepsilon}) = \max \{c.x \mid x \in F(B|x_j=\varepsilon)\}$

D'après ce qui précède , le test suivant permet de fixer la variable x_j:

$$(\quad v(B_j^{\varepsilon}) \leq \underline{v}(B) \ ) \quad => \quad x_j = 1-\varepsilon$$

Mais ce test est aussi complèxe dans sa mise en œuvre (résolution de B_j^{ε}) que la résolution du problème initial . Nous utiliserons donc un majorant de $v(B_j^{\varepsilon})$ pour envisager le test suivant , qui implique le précédent :

$$(\quad \overline{v}(B_j^{\varepsilon}) \leq \underline{v}(B) \ ) \quad => \quad x_j = 1-\varepsilon$$

Nous nous proposons de calculer des majorants de $v(B_j^{\varepsilon})$ de diverses manières : ces valeurs seront obtenues par résolution de relaxations de (B_j^{ε}) ou de(B).

2- LES TESTS CLASSIQUES

Au cours de ce paragraphes , nous allons présenter les tests classiques que l'on rencontre dans la littérature ([FA1],[FA2]) . Ces tests utilisent les principes exposés précédemment . Ces tests sont mis en oeuvre en séquentiel

Nous considérons la relaxation composite B°(w) du problème (B)

B°(w)	max cx	
	s.c. $wAx \leq wb$	$(w \in R_+^m)$
	$x \in \{0,1\}^n$	

1- Test V1

Effectuons une relaxation lagrangienne de ce knapsack B°(w) . Nous savons que si $v(RL(\lambda,B°(w)) \mid x_j = \varepsilon) < \underline{v}(B)$ alors x_j peut être fixée à $1-\varepsilon$.

Nous appliquons les tests exposés dans la relaxation Lagrangienne des knapsacks au problème B°(w) .

Nous savons que $v(RL(\lambda,B°(w)) \mid x_j = \varepsilon) = v(RL(\lambda,B°(w)) - |c_j - \lambda wA^j|$ avec $\varepsilon = 0$ si $(c_j-\lambda wA^j)$ est positif ($j \in U(\lambda,K)$) et 1 sinon .

Nous effectuons donc le test suivant :

Test V1
soit λ un réel positif , soit x_j une variable
En posant $U(\lambda,B°(w)) = \{ i \in \{1,...,n\} \mid c_i - \lambda wA^j > 0 \}$
si $\lambda wb + \sum_{l \in U(\lambda,B°(w))} (c_i-\lambda wA^i) - |c_j - \lambda wA^j| \leq \underline{v}(B)$
alors la variable x_j est fixée à 1 (resp. 0) si $(c_j - \lambda wA^j)$ est positif (resp. negatif) .

Remarque : Les multiplicateurs utilisés sont les c_j/wA^j ($j = 1,...,n$) et plus particulièrement le multiplicateur optimal λ^* .

Remarque : Il est à noter que la variable d'indice i tel que $\lambda = c_i / wA_i$ ne peut pas être éliminée par ce test car le coût réduit associé à cette variable est nul .

2- Test V2

Ce test peut permettre de fixer la variable d'indice i^* (telle que $\lambda^* = c_{i^*}/a_{i^*}$) qui n'a pas pu être fixée précédemment si le multiplicateur utilisé était λ^*. Pour cela nous considérons deux relâchements de (B°(w)) en fonction de la valeur prise par x_{i^*} .

V2 Si $v(\overline{B°(w) \mid x_{i^*} = \varepsilon}) \leq \underline{v}(B)$ alors x_{i^*} est fixée à $1-\varepsilon$.

3- Test V3

Dans le cas d'un echec du test précédent , nous envisageons l'exploration du premier niveau d'une arborescence en étudiant le problème (B°(w)) en fonction de la valeur prise par x_{i^*} .

Soient les deux problèmes suivants :
(B1) $(B|x_{i^*}=0;x_j=\varepsilon)$ et (B2) $(B|x_{i^*}=1;x_j=\varepsilon)$
Nous savons que $v(B|x_j=\varepsilon) = \max (v(B1) , v(B2))$
Donc si $\max (v(B1),v(B2)) < \underline{v}(B)$, x_j peut être fixée à $1-\varepsilon$.

En fait nous considérons bien sûr des relaxations Lagrangiennes de (B1) et (B2) associées aux multiplicateurs optimaux .
Nous calculons $v((B°(w)|x_{i^*}=\eta) ;x_j=\varepsilon)$ (η vaut 0 ou 1)
Soit λ_η le multiplicateur optimal associé au problème $(B°(w)|x_{i^*}=\eta)$
Soit $d_{\eta,j} = c_j - \lambda_\eta w A^j$ le coût réduit associé à la variable j dans le problème $(B°(w)|x_{i^*}=\eta)$

TEST V3 :
Si $\max \{v(\overline{B°(w)|x_{i^*}=0}) - |d_{0,j}| , v(B°(\overline{w)|x_{i^*}=1}) - |d_{1,j}|\} \leq \underline{v}(B)$
alors la variable x_j peut être fixée à 1 (resp 0) si $d_{\eta,j}$
(η=0,1) est positif (resp. négatif) .

4- Test V4

Nous envisageons de fixer certaines variables en considérant , non plus , une relaxation lagrangienne de B°(w) mais le p rogramme linéaire associé .

V4 : Si $v(B°(w) \mid x_j = \varepsilon) \leq \underline{v}(B)$ alors x_j peut être fixée à $1-\varepsilon$

5- Coûts des différents tests

Déterminons le coût de chacun de ces tests , utilisés successivement , pour une seule application du test . Nous pouvons calculer ce coût en fonction du nombre d'appels du programme de résolution d'un knapsack en continu N.K.R.

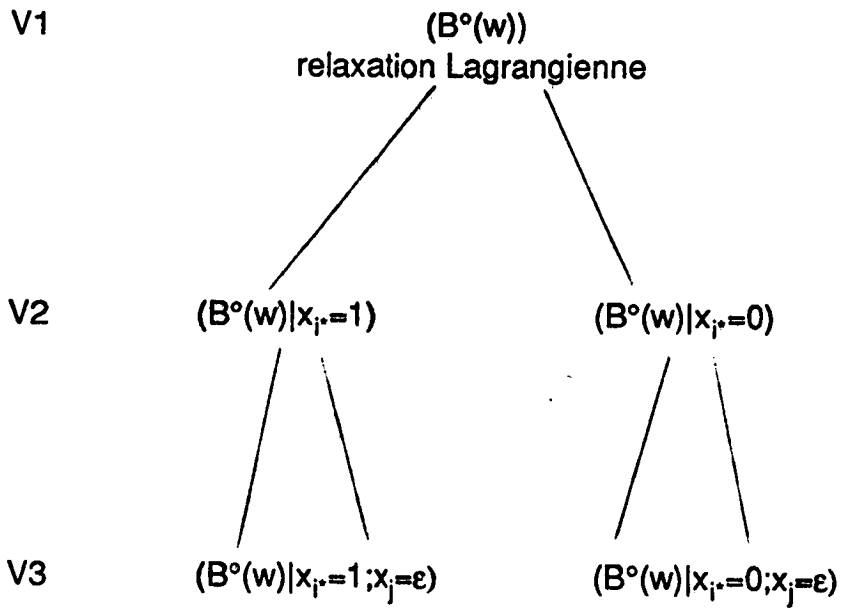
V1 : 1 appel de N.K.R. si on détermine l'indice i* de la variable de base
+ O(n-1) (calcul des coûts réduits) pour chaque multiplicateur considéré
= O(n-1)

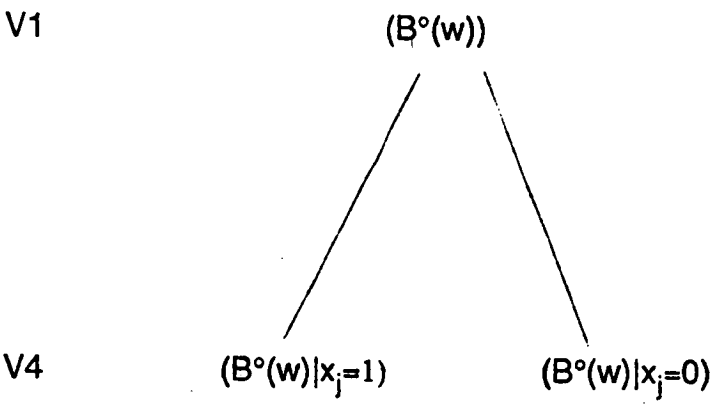
V2 : 2 appels de N.K.R.

V3 : aucun appel de N.K.R. , il suffit de calculer les coûts réduits puisque les valeurs de λ_1 et λ_2 ont été calculées par V2
+ 2O(n-1) (calcul des coûts réduits)
= O(n-1)

V4 : 1 appel de N.K.R. pour chaque variable traitée
$O(n^2)$ si on traite toutes les variables

6- Schéma représentant les problèmes outils utilisés





III- ELIMINATION DES CONTRAINTES

Nous nous proposons d'éliminer les contraintes dont l'absence ne modifie pas le problème . Ces contraintes seront dîtes contraintes redondantes .

Définition

Soient t une contrainte et S_t l'ensemble des vecteurs de $\{0,1\}^n$ qui vérifient les contraintes 1,...,t-1,t+1,...,m .
Nous savons que $S_t \supset F(B)$.

Une contrainte t est dîte redondante si et seulement si S_t et F(B) sont identiques .

Propriété : si une contrainte t est redondante tout vecteur de S_t est solution admissible de B . Pour qu'un vecteur soit solution de B , il suffit qu'il vérifie les contraintes 1,...,t-1,t+1,...,m . La contrainte t peut donc être éliminée .

Corollaire 1
Soit une contrainte t . Si tous les vecteurs de $\{0,1\}^n$ vérifiant une combinaison linéaire positive des autres contraintes vérifient aussi la contraintes t , alors la contrainte t est redondante et peut être éliminée.

Si $(\forall x \in \{0,1\}^n \quad wAx \leq wb \Rightarrow A_t x \leq b_t)$ alors t est redondante
avec $w \in R_+^m$ et $w_t = 0$

Corollaire 2
Soit $(B^t(w))$ le problème suivant :

$$\max A_t x$$
$$wAx \leq wb$$
avec $w \in R_+^m$ et $w_t = 0$

Si la valeur de $B^t(w)$ est inférieure à b_t alors la contrainte t est redondante et peut être éliminée .

Toujours pour les mêmes raisons de complexité , nous utilisons un majorant de cette valeur .

Nous éliminerons donc les contraintes t telles que
$$\bar{v}(B^t(w)) \leq b^t \quad \text{avec} \quad w \in R_+^m \text{ et } w_t = 0$$

Nous nous proposons de calculer des majorants de $v(B^t(w))$ en considérant des relaxations de ce problème .

2- LES TESTS CLASSIQUES

Au cours de ce paragraphes , nous allons présenter les tests classiques que l'on rencontre dans la littérature ([FA1],[FA2]) . Ces tests utilisent les principes exposés précédemment .

Comme nous l'avons vu précédemment , nous considérons le problème

$(B^t(w))$ $\max A_t x$
s.c. $wAx \leq wb$
$x \in \{0,1\}^n$
avec $w \in R_+^m$, et $\exists i \neq t$ $w_i \neq 0$

Si $v(B^t(w)) \leq b_t$ alors la contrainte t peut être éliminée .
Nous envisageons diverses manières de calcul de ce majorant .

1- Test C1

Ce premier test utilise le programme linéaire associé à $B^t(w)$

C1 Si $v(\overline{B^t(w)}) \leq b_t$ alors la contrainte t peut être éliminée .

2- Test C2

Dans le cas d'un échec du test précédent , nous explorons le premier niveau d'une arborescence en fonction de la valeur prise par la variable de base d'indice i^* du problème $B^t(w)$.

Soit $v2 = \max \{ (v(\overline{B^t(w)|x_{i^*}=0})) , (v(\overline{B^t(w)|x_{i^*}=1})) \}$

Si $v2 \leq b_t$ alors la contrainte I peut être éliminée .

3- Test C3

Si le test précédent échoue , et si l'une des deux valeurs $v(B^t(w)|x_{i^*}=1-\varepsilon)$ (ε= 0 ou 1) est inférieure à b_t , alors nous pouvons essayer d'améliorer l'autre valeur . Pour cela , comme précédemment , nous fixons la variable de base $x_{i^*(\varepsilon)}$ du problème $(B^t(w)|x_{i^*}=\varepsilon)$.

Soit $v3=\max\{v(B^t(w)|x_{i^*}=\varepsilon;x_{i^*(\varepsilon)}=0),v(B^t(w)|x_{i^*}=\varepsilon;x_{i^*(\varepsilon)}=1)\}$
Si $v3 \leq b_t$ alors la contrainte t peut être éliminée .

4- Coût des différents tests

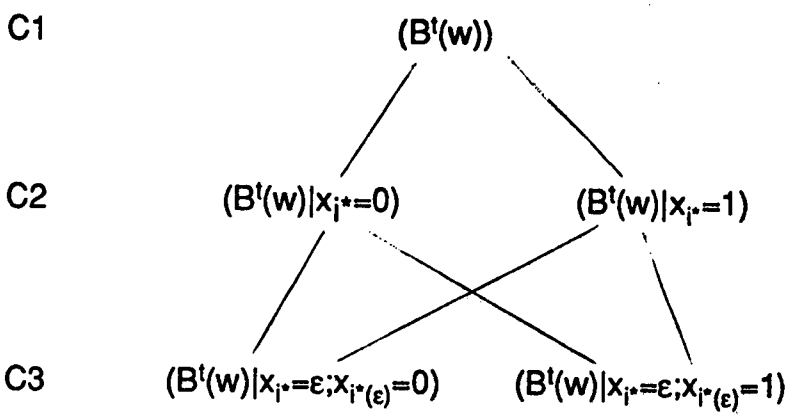
De même que pour l'élimination des variables , nous calculons le coût d'un test en fonction du nombre d'appels du programme de résolution du knapsack en continu (relâchement) N.K.R. .

C1 : 1 appel

C2 : 2 appels

C3 : 2 appels

5- Schéma représentantant les problèmes outils utilisés



CHAPITRE 3 : ETUDE D'UNE NOUVELLE MISE EN OEUVRE DE L'ALGORITHME PR^2

Une première implantation de l'algorithme PR^2 a été effectuée en 1987 (mémoire d'ingénieur de l'IIE [VA]) . Nous allons présenter les caractéristiques de ce programme ($PR^2$87) puis nous proposerons une nouvelle mise en œuvre ($PR^2$88) .

I- RAPPEL DU PRINCIPE DE PR^{2}87

Ce programme est un premier pas vers la mise en œuvre de PR^2 . Nous allons présenter son principe général , plus particulièrement les tests utilisés pour les éliminations , puis nous le comparerons avec le programme FP83 (Freville et Plateau) , enfin nous présenterons l'aspect parallèle du programme . Ce programme a été implanté sur une machine multiprocesseurs assynchrone à mémoire commune (CRAY 2) . Vous trouverez une description de la machine en annexe .

1- Principe général

Les tests utilisés dans PR^{2}87 sont plus simples que ceux présentés dans PR^2 . Les ensembles W_1 et W_2 sont réduits à l'ensemble des vecteurs canoniques de R^m . Seuls les problèmes (B°_i) et (B^l_i) seront pris en compte. De plus seuls les tests simples V1 et C1 sont utilisés . La limitation aux seuls problèmes (B°_i) et (B^l_i) et aux tests V1 et C1 est compensée par une exploration plus étendue des possibilités d'éliminations.

Dans les algorithmes précédents seul le multiplicateur Lagrangien optimal λ^* était utilisé . Or , nous avons vu que les autres multiplicateurs , éléments de l'ensemble $\{c_k/a_k \mid k=1,\ldots,m\}$, peuvent fournir de meilleurs majorants . Les valeurs associées à chaque multiplicateur sont calculées, soient

$\forall \lambda \ \{c_k/a_k \setminus k=1,\ldots,n\}$

$\forall i \in \{1,\ldots,m\}$, $\forall j \in \{1,\ldots,n\}$ $\quad v(RL(\lambda,B^\circ_i)|x_j=\varepsilon)$

Si $v(RL(\lambda,B^\circ_i)|x_j=\varepsilon) \leq \underline{v}(B)$ alors la variable j est fixée à $1-\varepsilon$

Dans les algorithmes précédents chaque contrainte n'était comparée qu'avec une seule contrainte . Ici , chaque contrainte est comparée avec toutes les autres contraintes non elimiminées . Tous les knapsacks (B^t_i) tels que i est différent de t et la contrainte i n'a pas été éliminée sont traités .

Principe des éliminations

Eliminations triviales par les tests R1 , R2 et R3

Elimination des variables
Répéter
| Nombre de variables éliminées = 0
| Faire pour chaque contrainte i non éliminée
| | Faire pour chaque multiplicateur $\lambda=c_k/a_k$
| | | Appliquer le test V1 à chaque variable x_j non éliminée
| | Fait
| Fait
Jusqu'à nombre de variables éliminées = 0

Elimination des contraintes
Pour chaque contrainte i non éliminée
| Pour chaque contrainte k non éliminée ($k\neq i$)
| | Appliquer C1 sur (B^k_i)
| Fait
Fait

2- Comparaison avec les tests réalisés par l'algorithme FP83

Le tableau suivant résume les différences entre les deux algorithmes .

	FP83	PR^2_{87}
	ELIMINATION DES VARIABLES	
problèmes utilisés	(B°_i) $(B^\circ(w))$ w optimal	(B°_i)
Tests	V1 , V2 , V3	V1
λ	λ^* optimal	tous les multiplicateurs
2ème phase	V4	
	ELIMINATION DES CONTRAINTES	
problèmes utilisée	(B^l_i) tq w_i maximal $(B^k(w))$	tous les (B^l_i)
Tests	C1 , C2 , C3	C1

II- L'ALGORITHME PR²88

L'algorithme PR²88 représente une deuxième étape vers la mise en œuvre de PR² . Nous allons modifier PR²87 en fonction des résultats obtenus ; dans le but d'obtenir une réduction encore meilleure . Nous allons comparer les résultats obtenus par les deux programmes FP83 et PR²87 et en déduire plusieurs scénari d'évolution de PR²87 .

1- Comparaison des résultats de PR²87 et FP83

Les deux programmes ont été testés sur des problèmes tests de la littérature (voir page suivante) ([VA],[PR1]).

Tableau des résultats

Problème : nom du problème
Taille : taille du problème d'origine : nombre de contraintes x nombre de variables
FPR 83 : taille du problème réduit après exécution du programme séquentiel FPR 83
PR^2 87 : taille du problème réduit après exécution du programme parallèle PR^2 87
temps $PR^2$87 : temps d'exécution pour le programme PR^2 87 en secondes

Problème	Taille	FPR 83	PR^2 87	temps PR^2 87
Pet 1	10 x 6	0 x 0	0 x 0	0,018
Pet 2	10 x 10	1 x 3	1 x 3	0,024
Pet 3	10 x 15	5 x 8	5 x 8	0,038
Pet 4	10 x 20	2 x 8	2 x 10	0,043
Pet 5	10 x 28	2 x 9	2 x 10	0,055
Pet 6	5 x 39	4 x 28	5 x 38	0,045
Pet 7	5 x 50	4 x 38	5 x 46	0,068
NW 1	2 x 28	1 x 4	1 x 4	0,031
NW 2	2 x 28	1 x 5	2 x 11	0,028
NW 3	2 x 28	2 x 19	2 x 21	0,025
NW 4	2 x 28	1 x 5	1 x 5	0,031
NW 5	2 x 28	0 x 0	0 x 0	0,031
NW 6	2 x 28	2 x 7	2 x 6	0,029
NW 7	2 x 105	2 x 12	2 x 61	0,106
NW 8	2 x 105	2 x 29	2 x 102	0,070
HP 1	4 x 28	4 x 27	4 x 28	0,023
HP 2	4 x 35	4 x 34	4 x 34	0,033
ST 1	30 x 60	30 x 40	30 x 53	0,574
ST 2	30 x 60	30 x 37	30 x 51	0,556

Nous rappelons que FP83 utilise des tests plus élaborés , mais qu'il considère seulement le multiplicateur Lagrangien optimal . Néanmoins , PR^2 87 est au moins aussi bon pour quinze problèmes (avec une amélioration pour sept d'entre eux).

La dernière colonne détaille les temps CPU en millisecondes pour le programme PR^2 87 implémenté sur le CRAY2 travaillant avec un seul processeur . Ceci permet de montrer que ,même la simulation du parallèlisme conduit à de très petits temps de calcul : de 0,180 s (pour le problème de Petersen 1 , 10x6 réduit à 0x0) à 0,570 s (pour le problème de Senju et Toyoda 1 , 30x60 réduit à 30x53) . D'autre part , ces temps comprennent la lecture des données qui représente environ 30 pour cent du temps global , et peuvent différer d'une exécution à l'autre du fait du non déterminisme ([PR1]).

Des expériences supplémentaires ont été effectuées sur le CRAY2 afin de mesurer l'impact de l'utilisation de plus d'un processeur . Dans le tableau ci-dessous , pour les deux problèmes de Senju et Toyoda , nous indiquons les valeurs des paramètres suivants :

T_j : le temps avec j processeurs (calculé d'après les informations fournies par la machine , c'est-à-dire les temps CPU t_i utilisés lorsque la machine travaille avec i processeurs)

$$T_j = \Sigma_{i=1,\ldots,j}\, t_i$$

S_j : le "speed-up" expérimental avec j processeurs mesuré par le quotient T_1/T_j

Taille	j	T_j (ms)	S_j
30 x 60	1	632	1
	2	307	1,50
	3	277	1,67
	4	338	1,33
30 x 50	1	66	1
	2	332	1,33
	3	336	
	4	558	0,33

Ces exemples montrent , que , pour des problèmes de petites tailles , le temps d'activation , de création , de synchronisation et d'accès à la mémoire partagée ne doit pas être négligé . Il est clair que l'utilisation de quatre processeurs sera plus intéressant pour des problèmes ayant un nombre de variables plus élevé que ceux de la littérature . Mais nous devons reconsidérer les problèmes de synchronisation et de longueur des tâches . Le travail effectué par chaque tâche semble être trop peu volumineux (il arrive même , que la première tâche ait terminé la totalité du travail avant que les autres n'aient commencé) .

Conclusion

Les résultats de cette première expérience indiquent que :
- Des tests de réduction plus élaborés doivent être implémentés afin d'augmenter le nombre de variables éliminées . En outre , une augmentation du travail alloué à chaque processeur devrait conduire à une meilleure exploitation de l'ordinateur , par une meilleure répartition du travail entre les différentes tâches .
- De bons multiplicateurs composites w devront être générés afin de construire de meilleures relaxations du problème .

Nous allons étudier l'influence de ces modifications sur la phase de réduction (introduction des tests V2,V3,V4,C2,C3 , génération de multiplicateurs composites , modification de la synchronisation des tâches au cours de l'élimination des variables) .

2- Nouvel algorithme de fixation des variables

a) Génération de multiplicateurs composites

Le programme FP83 calcule le meilleur multiplicateur composite . Ce multiplicateur w^* est obtenu par un algorithme itératif dit de "sous-gradient" (en fait cet algorithme fournit le meilleur multiplicateur lagrangien . Des méthodes dites de "quasi sous-gradient" permettent de calculer le meilleur multiplicateur composite mais convergent beaucoup moins vite que les précédentes . Nous considérons que le multiplicateur fourni par l'algorithme de sous-gradient est satisfaisant pour notre problème) . Le programme du sous-gradient construit une suite de multiplicateurs (w_k) dont w^* est la limite .

w^* est solution du problème suivant

(D_L) $\min_w v(w)$

$w \in R_+^m$

où $v(w)$ est la valeur de la relaxation Lagrangienne de (B) de paramètre w : $v(w) = \max_x (c.x. + w(b - Ax))$ $x \in \{0,1\}^n$

Le programme FP83 applique les tests de réduction (élimination des variables) , non seulement aux problèmes $(B°_i)$ et mais aussi au problème $(B°(w^*))$.

D'autre part , le parallèlisme nous permet de considérer un nombre beaucoup plus important de knapsack . Nous envisageons donc d'utiliser le knapsack $(B°(w^*))$, et en plus , les knapsack $(B°(w_k))$, où la suite (w_k) est une sous-suite de la suite itérative fournie par la programme de sous-gradient .

Deux problèmes apparaissent dans le choix de la sous-suite :

- les différents w_k doivent fournir des knapsack outils assez différents . Il est inutile de considérer deux valeurs de w si ces deux multiplicateurs éliminent les mêmes variables . Nous dirons donc que les w_k doivent être assez éloignés les uns des autres .
- Les indices des w_k doivent être assez élevés pour que leurs valeurs soient significatives .

Nous avons choisi de générer deux suites $(W°_k)$ et (w^1_k) qui sont obtenues avec des multiplicateurs initiaux différents :

$w°_0 = (0,\ldots,0)$ $\quad w^1_0 = (1,\ldots,1)$

et de ne retenir qu'un seul multiplicateur sur cinq multiplicateurs générés .

Algorithme de calcul des multiplicateurs composites

Soient $(W^i_k)_k$ les éléments de la suite générée par l'algorithme (i=0,1)
$(w_k)_k$ les multiplicateurs composites retenus , qui génèreront des knapsacks outils

Début
iter := 0 ; (* indice de la suite W^i_k *)
indice := 0 ; (* indice de la suite w_k *)
Pour i de 0 à 1 Faire
Si i = 0 alors $W^i_0 = (0,...,0)$ Fsi ;
Si i = 1 alors $W^i_0 = (1,...,1)$ Fsi ;
Arrêt := faux ;
Tant que non (Arrêt) Faire
iter := iter + 1 ;
Calculer W^i_{iter} (sous-gradient) ;
Si iter est un multiple de 5 alors
indice := indice + 1 ;
$w_{indice} := W^i_{iter}$;
Fsi ;
Test d'arrêt : arrêt si $\dfrac{\| W^i_{iter} - W^i_{iter-1} \|}{\|W^i_{iter}\| \times \|W^i_{iter-1}\|} \leq \varepsilon$
Si Arrêt alors
Si indice = 0 alors
indice := 1 ;
$w_{indice} := W^i_{indice}$;
Fsi ;
iter := 0 ;
Fsi ;
Fait ;
Fait ;
Fin

Nous obtenons une suite $(w_k)_{k=1,indice}$. Nous avons (m + indice) knapsacks outils , qui seront traités dans l'ordre suivant (des multiplicateurs) :

$$w_{indice} , \underbrace{e_1 , \dots , e_m}_{\text{base canonique}} , w_1 , \dots , w_{indice-1}$$

b) Nouveaux tests

Nous avons donc envisagé d'introduire les tests V2,V3 et V4 . Mais nous avons pu montrer que les tests V2 et V4 ne peuvent apporter aucune élimination supplémentaire du fait que , lors du tests V1 , tous les multiplicateurs lagrangiens sont traités .

Démonstration

Soit x_j une variable . Montrons que si V3 (ou V5) permet d'éliminer cette variable , alors V1 peut aussi l'éliminer . V1 étant le test le moins coûteux , il sera donc inutile d'effectuer V3 (ou V5) .

Hypothèse : $v(B°_i \mid x_j = \varepsilon) \leq \underline{v}(B)$

Nous voulons montrer que $\exists \lambda \in \{c_k/a_{ik} \mid k=1,\ldots,n\}$ tel que

$$v(RL(\lambda , B°_i) \mid x_j = \varepsilon) \leq \underline{v}(B)$$

D'après la propriété 1 du chapitre 3 (II) appliquée à $(B°_i \mid x_j = \varepsilon)$

$\exists \lambda \in \{c_k/a_{ik} \mid k=1,\ldots,n\}$ tel que

$$v(RL(\lambda , B°_i) \mid x_j = \varepsilon) = v(B°_i \mid x_j = \varepsilon)$$

Donc , si l'hypothèse est vérifiée alors

$$v(RL(\lambda , B°_i) \mid x_j = \varepsilon) \leq \underline{v}(B)$$

La variable a donc pu être éliminée par V1 , par le multiplicateur λ .

En revanche , nous pouvons montrer que le test V3 peut donner des éliminations supplémentaires .

Les tests suivants sont appliqués à tous les multiplicateurs composites (w_k et e_k) générés précédemment .

Rappelons le principe des tests de fixation des variables

Soient x_j une variable , $\varepsilon \in \{0,1\}$ et w un multiplicateur composite .
Soit $v(B°(w) \mid x_j = \varepsilon)$ un majorant de $v(B°(w) \mid x_j = \varepsilon)$.

Si $v(B°(w) \mid x_j = \varepsilon) \leq \underline{v}(B)$ alors x_j peut être fixée à $1-\varepsilon$.

Le problème revient à trouver de bonnes valeurs de $v(B°(w) \mid x_j = \varepsilon)$.

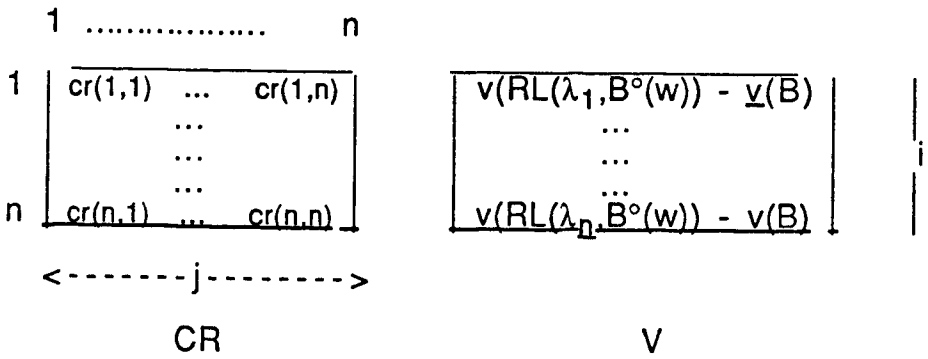
α) Relaxations Lagrangiennes : Test V1

Rappelons le principe de ce test

Soit $\lambda_i \in \{ c_i / wA^i \mid i \in \{1,...,n\} \}$ un multiplicateur Lagrangien .
Soit cr(i,j) le coût réduit associé au multiplicateur λ_i et à la variable x_j $\quad cr(i,j) = c_j - \lambda_i(wA^i)$

Si $|cr(i,j)| \geq v(RL(\lambda_i , B°(w))) - \underline{v}(B)$ alors x_j peut être fixée à 1 (si cr(i,j) > 0) , à 0 (si cr(i,j) < 0) .

où $v(RL(\lambda_i , B°(w))) = \lambda_i(wb) + \sum_{j=1,...,n} cr(i,j) \mid cr(i,j)>0$



Notons CR la matrice des coûts réduits et V la matrice colonne des $v(RL(\lambda_i,B°(w)) - \underline{v}(B)$. Il s'agit de comparer chacun des éléments de CR , cr(i,j) aux éléments de V , V(i) .

Si $|cr(i,j)| \geq V(i)$ alors la variable x_j peut être fixée .

PR2 87 effectue ce test pour toutes les variables et tous les multiplicateurs .
Nous conservons ce test , mais en cas d'échec nous essayons d'améliorer la valeur de $\bar{v}(B°(w) \mid x_j = \varepsilon)$.

β - Test dît "des relations binaires"

Principe

Soit une variable x_j qui n'a pas pu être fixée par le test précédent .

$$\forall i \in \{1,...,n\} \quad |cr(i,j)| < V(i)$$

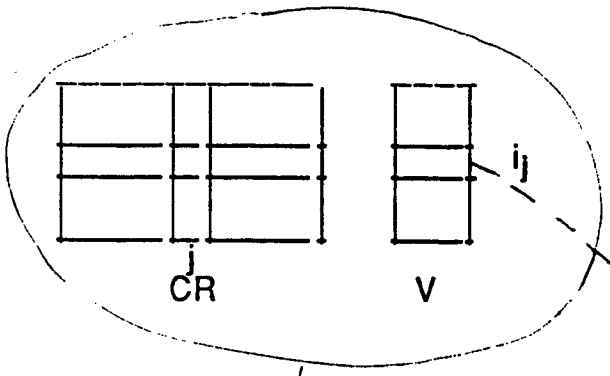
Soit $i_j \in \{1,...,n\}$ tel que $V(i_j) - |cr(i_j,j)| = \min \{ V(i)-|cr(i,j)| \mid i \in \{1,...,n\} \}$
Le multiplicateur λ_{i_j} sera considéré comme étant le "meilleur" multiplicateur pour la variable x_j .
Si $cr(i_j,j) > 0$, nous posons $\varepsilon_j = 0$, sinon $\varepsilon_j = 1$.

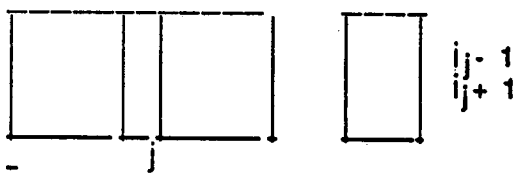
Nous allons essayer d'améliorer la valeur de $v(B°(w) \mid x_j=\varepsilon_j)$.

Nous allons simuler la fixation de x_j à ε_j , et étudier les conséquences sur les autres relaxations ($i \neq i_j$) . Fixer x_j à ε_j peut nous conduire à fixer d'autres variables (nous noterons X^{η}_j les ensembles des variables ainsi fixées à η , $\eta \in \{0,1\}$) . Puis nous étudierons les effets de ces fixations sur la relaxation d'indice i_j .

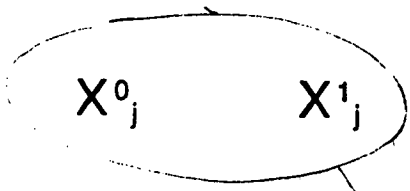
Nous calculerons $v(RL(\lambda_{i_j} , B°(w)) \mid x_j=\varepsilon_j , \forall x_k \in X^0_j \; x_k=0 , \forall x_k \in X^1_j \; x_k=1)$
Puisque certaines variables sont fixées , nous pouvons penser que cette dernière valeur sera inférieure strictement à la valeur calculée par V1, et qu'elle peut devenir inférieure au minorant de la valeur du problème (B) et que nous pourrons fixer x_j à $1-\varepsilon_j$.

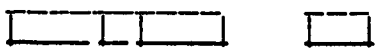
Principe du test des relations binaires





Simulation de la fixation de x_j à ϵ_j sur les autres relaxations . Cette fixation entraîne la fixation d'autres variables .





Relaxation i_j

Etude des conséquences de la fixation des variables de X^0_j et X^1_j à 0 et à 1 sur la relaxation Lagrangienne i_j .
x_j peut-il être fixé à $1-\epsilon_j$.

Rappels

Afin de faciliter la compréhension de l'exposé nous rappelons le principe de la résolution des knapsacks , et des relaxations Lagrangiennes des multiknapsacks .

Soit $i \in \{1,...,n\}$, $i \neq i_j$. Considérons la relaxation Lagrangienne associée au multiplicateur λ_i

$$v(RL(\lambda_i, B^\circ(w)) = \text{Max}_x \ (c.x + \lambda_i(wb - wAx))$$
$$= \lambda_i wb + \sum_{k=1,...,n} ((c_k - \lambda_i wA^k) \quad \text{positifs})$$

coût réduit associé à la variable k et à la relaxation i cr(i,k)

(pour maximiser la valeur de ce problème il suffit de poser égales à 1 les variables associées à un coût réduit positif) .

Simulation de la fixation de x_j à ε_j

Simulons la fixation de la variable x_j à ε_j dans les relaxations d'indice i différent de i_j .

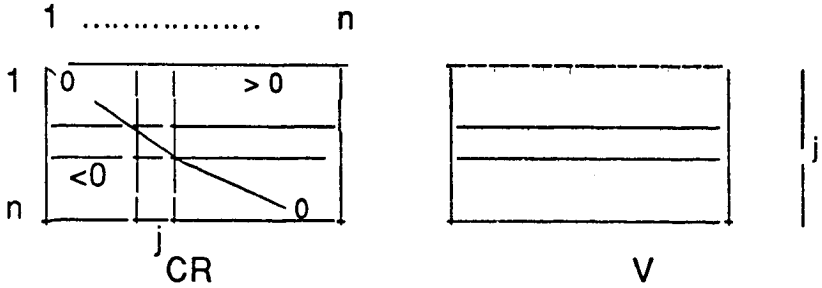
Si ($\varepsilon_j = 1$ et $cr(i,j) < 0$) ou ($\varepsilon_j = 0$ et $cr(i,j) > 0$) , nous n'avons plus la solution optimale pour la relaxation d'indice i et nous devons soustraire le coût réduit $|cr(i,j)|$ à la valeur de la relaxation et donc à V(i) .

Illustrons ceci à l'aide des matrice CR et V .

Supposons que les variables sont ordonnées de telle sorte que :

$$\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$$

Alors les coûts réduits des se trouvant au-dessus de la diagonale de CR sont positifs et ceux se trouvant au-dessous de la diagonale sont négatifs .



En conséquence , si la variable x_j est fixée à 0 (respectivement à 1) , les lignes de la matrice V situées au-dessus de la ligne i* seront modifiées (respectivement les lignes situées au-dessous) . Il est à noter qu'au cours de ces opérations la ligne de la relaxation i_j n'est pas modifiée .

Les valeurs de V ayant été modifiées , et ayant été diminuées , cette matrice contient maintenant les valeurs $v(RL(\lambda_i, B°(w) | x_j = \varepsilon_j) - \underline{v}(B)$. Il se peut que certains coûts réduits (correspondant à des relaxations différentes de i_j) soient devenus , en valeur absolue , inférieurs à la valeur de V correspondante et que l'on puisse fixer d'autres variables . Nous réitérons le procéssus jusqu'à ce que nous ne puissions plus fixer de variables . Nous avons ainsi construit deux ensembles X^1_j et X^0_j de variables fixées à 1 ou à 0 , sous l'hypothèse que x_j est fixée à ε_j .

Conséquences de la fixation de x_j à ε_j sur la relaxation i_j

Si l'un au moins des deux ensembles X^n_j est non vide , nous étudions les conséquences de l'hypothèse x_j fixée à ε_j sur la relaxation associée à i_j.

Pour chaque variable k de X^0_j , si $cr(i_j,k)>0$, alors $V(i_j) = V(i_j) - cr(i_j,k)$
Pour chaque variable k de X^1_j , si $cr(i_j,k)<0$, alors $V(i_j) = V(i_j) + cr(i_j,k)$

$V(i_j)$ peut ainsi devenir inférieure à $|cr(i_j,j)|$, ce qui signifie alors que

$v(RL(\lambda_{i_j} , B°(w)) \mid x_j=\varepsilon_j , \forall x_k \in X^0_j \; x_k=0 , \forall x_k \in X^1_j \; x_k=1) \leq \underline{v}(B)$

et x_j peut être fixée à $1- \varepsilon_j$.

Algorithme

Cette algorithme explicite les opérations effectuées lors de l'application du test des relations binaires à une variable x_j . Les matrices CR et V sont supposées données .

Début
Détermination de l'indice i_j , tel que : $V(i_j) - |cr(i_j,j)| = \min_i (V(i) - |cr(i,j)|)$
Si ($cr(i_j,j) > 0$) alors $\varepsilon_j = 0$ Fsi ;
Si ($cr(i_j,j) < 0$) alors $\varepsilon_j = 1$ Fsi ;
Simulation de la fixation de la variable x_j à ε_j
$X^0_j := \emptyset$; $X^1_j := \emptyset$;
Répéter
Mettre à jour les valeurs de la matrice V en fonction des variables fixées ;
Pour chaque multiplicateur λ_i Faire $(i \neq j)$
Pour chaque variable x_k qui a été fixée Faire
Si (((cr(i,k)>0) et (x_k fixée à 0)) ou
((cr(i,k)<0) et (x_k fixée à 1)) alors
V(i) := V(i) - |cr(i,k)|
Fsi ;
Eliminer la ligne k de V ;
Eliminer la ligne k et la colonne k de CR ;
Fait ;
Pour chaque variable x_l non fixée Faire
Pour chaque ligne i de V Faire $(i \neq j)$
Si $|cr(i,l)| \geq V(i)$ alors
Si cr(i,l) > 0 alors fixer x_l à 1 ; $X^1_k := X^1_k \cup \{x_l\}$ Fsi ;
Si cr(i,l) < 0 alors fixer x_l à 0 ; $X^\circ_k := X^\circ_k \cup \{x_l\}$ Fsi ;
Fsi ;
Fait ;
Fait ;
Appliquer le test R1 au problème réduit ;
Fait ;
Jusqu'à aucune variable ne peut plus être fixée ;
Modifier $V(i_j)$ en fonction des ensembles X^0_j et X^1_j ;
Si $|cr(i_j,j)| \geq V(i_j)$ alors
x_j peut être fixée à $1-\varepsilon_j$
Fsi ;
Fin

γ) Conclusion

Au cours de la phase de fixation des variables , nous effectuons donc le test V1 , appliqué à toutes les variables et tous les multiplicateurs Lagrangiens . Puis , nous appliquons le test des relations binaires à toutes les variables non éliminées précédemment .

Une première expérience a montré que le test V4 apporte peu de fixations supplémentaires , par rapport au test V1 . Nous avons décidé de supprimer ce test , qui sera remplacé par le test des relations binaires .

Nous allons détailler le traitement effectué lors de la fixation des variables sur un knapsack outil ($B^\circ(w)$) (w peut être un élément des suites engendrées par l'algorithme de sous-gradient ou un élément de la base canonique)

Algorithme du test de fixation des variables appliqué à un knapsack outil $B°(w°)$

```
Début

    Tests de réductions triviales ;
    Initialisation des données : c , wA et wb ;
    Initialisation des matrices CR et V ;

    Répéter
    |
    |   Application du test V1 à toutes les variables avec tous les
    |   multiplicateurs Lagrangiens ;
    |
    |   Mise à jour des matrices CR et V en fonction des fixations
    |   effectuées par le test V1 :
    |     Pour toutes les variables fixées xj Faire
    |     |   Pour toutes les lignes i de la matrice V Faire
    |     |     Si ( ( ( xj fixée à 0 ) et ( cr(i,j) > 0 ) )   ou
    |     |     |      ( ( xj fixée à 1 ) et ( cr(i,j) < 0 ) )  alors
    |     |     |        V(i) := V(i) - |cr(i,j)|
    |     |     Fsi ;
    |     |   Fait ;
    |     |   Eliminer la ligne j de V ;
    |     |   Eliminer le ligne j et la colonne j de CR ;
    |     Fait ;
    |
    Jusqu'à V1 ne permet plus de fixer de variables ;

    Pour toutes les variables xj non fixées Faire
    |   Application du test des relations binaires à xj ;
    |   Si xj a été fixée alors
    |   |   Mettre à jour les matrices CR et V
    |   Fsi
    Fait

Fin
```

c- Etude de la modification de la synchronisation des tâches au cours de l'élimination des variables

Le problème de la synchronisation des tâches est très important pour l'efficacité du parallèlisme . Les tâches doivent se synchroniser assez souvent afin que chacune puisse profiter des résultats des autres . Mais la synchronisation est coûteuse . Il est donc nécéssaire de trouver un compromis . Ce problème se pose plus particulièrement au cours de l'élimination des variables , puisque nous avons vu que l'élimination des contraintes est asynchrone .

Nous avons vu que les tâches se synchronisent à la fin de chaque phase . Ecrivons l'algorithme de l'élimination des variables .

```
Tâche principale
Début
  Répéter
    Tant que des knapsacks outils n'ont pas été traitées faire
      Allouer les knapsack (B°(w)) aux tâches libres ;
    Fait ;
    Synchronisation des tâches ;
    Regroupement des résultats des tâches ;
    Mettre le problème à jour en fonction des éliminations
    effectuées ;
  Jusqu'à aucune élimination n'a été effectuée
Fin
```

```
Les autres tâches
Début
  Répéter
    Demander un knapsack (B°(w))
    Si un knapsack a été alloué alors
      Elimination des variables ;
      Mise à jour du problème local à la tâche en fonction des
      éliminations ;
    Fsi
  Jusqu'à il n'y a plus de knapsack
Fin
```

Nous avons décidé de synchroniser les tâches plus souvent . Lorsqu'une tâche a éliminé un certain nombre de variables elle peut mettre à jour un tableau contenant la liste des variables éliminées . Nommons X1 ce tableau. Nous devons minimiser le nombre d'accès à ce tableau afin de minimiser les attentes . De plus lorsqu'une tâche a mis à jour X1 , elle indique aux autres tâches qu'une modification a été effectuée en remplissant un tableau MAJ que les tâches lisent régulièrement . Les accès aux tableaux X1 et MAJ sont gérés par deux sémaphores .

```
Tâche principale

Début
    Initialiser X1 à aucune variable éliminée ;
    Répéter
        Tant que des contraintes n'ont pas été traitées faire
            Allouer les knapsack (B°(w)) aux tâches libres ;
        Fait ;
        Synchronisation des tâches ;
        Regroupement des résultats ;
        Mise à jour du problème et de X1 en fonction des
        éliminations ;
    Jusqu'à aucune variable n'a été éliminée
Fin
```

Les autres tâches

```
Début
    Recopier X1 dans une variable locale ;
    Répéter
        Demander un knapsack (B°(w)) ;
        Si un knapsack a été alloué alors
            Elimination des variables ;
            Mise à jour du problème local à la tâche en fontion des
            éliminations ;
            Si le nombre des éliminations est assez grand alors
                Demander l'accès à X1 en ouvrant le sémaphore
                associé
                Mettre X1 à jour ;
                Recopier X1 dans une variables locales (pour tenir
                compte des mises à jour éventuelles effectuées
                par les autres taches) ;
                Refermer le sémaphore associé à X1 ;
                Ouvrir le sémaphore associé à MAJ ;
                Mettre à jour MAJ pour indiquer aux autres tâches
                que des modifications ont eu lieu ;
                Fermer le sémaphore associé à MAJ;
            Sinon
                Demander l'accès au tableau MAJ ;
                Si des modifications ont eu lieu alors
                    Demander l'accès à X1 en ouvrant le sémaphore
                    associé ;
                    Recopier X1 dans une variable locale ;
                    Fermer le sémaphore associé à X1 ;
                Fsi ;
                Fermer le sémaphore associé à MAJ ;
            Fsi ;
        Fsi ;
    Jusqu'à aucune élimination n'a été effectuée par la tâche
Fin
```

3) Nouvel algorithme d'élimination des contraintes

L'algorithme PR^2 88 appliquait le test C1 à tous les knapsacks outils (B^i_k) , ce qui signifie que toutes les contraintes étaient comparées avec toutes les autres contraintes encore actives . Ceci a permis d'éliminer des contraintes supplémentaires , par rapport aux résultats du programme FPR 83 , mais celui-ci fournissait parfois de meilleurs résultats . C'est pourquoi , nous avons décidé d'apporter deux compléments :

- l'étude de knapsacks outils supplémentaires , knapsacks générés à partir des multiplicateurs composites (voir 1)

- l'introduction des tests C2 et C3 , qui en cas d'échec du test C1 considèrent les séparations des knapsacks sur leur variable de base . Nous rappelons que ces deux tests font appel à l'algorithme NKR de résolution des knapsacks en continu .

D'autre part , comparer chaque contrainte à toutes les autres contraintes et à tous les knapsacks outils générés par la sous-suite de multiplicateurs composites nous semble trop coûteux . C'est pourquoi nous limiterons ces comparaisons aux seuls multiplicateurs de la base canoniques (donc avec les autres contraintes) et avec le multiplicateur considéré comme étant optimal .

Présentons le nouvel algorithme d'élimination des contraintes . Soit une contrainte i , l'algorithme ci-dessous présente une adaptation séquentielle de l'étude de son élimination (ici la même tâche effectue tous les tests , en parallèle , il est possible que plusieurs tâches traitent plusieurs knapsacks outils relatifs à la contrainte i simultanément) . Nous supposons que le multiplicateur optimal w* a été calculé précédemment .

Nous avons donc un ensemble W de multiplicateurs générateurs de knapsacks outils $W := \{ w^* , e_1 , \dots , e_m \}$
et les knapsacks outils associés :

$(B^i(w^*))$ Max $A_i x$
s.c. $w^*Ax \leq wb$ pour w^*
$x \in \{0,1\}^n$

(B^i_j) Max $A_i x$
s.c. $A_j x \leq b_j$ pour e_j
$x \in \{0,1\}^n$

Algorithme

Début

W := { w^* , e_1 , ... , e_m } , ensemble des multiplicateurs générateurs de knapsacks outils ;

Pour chaque multiplicateur w de W Faire

Initialisation des données associées à ($B^i(w)$) ou à (B^i_j) ;

Appliquer le test C1 au knapsack outil ;

Si le test C1 échoue alors

Appliquer le test C2 au knapsack outil ;

Si le test C2 échoue alors

Appliquer le test C3 au knapsack outil ;

Fsi ;

Fsi ;

Fait

Fin

CHAPITRE 4 : L'ALGORITHME PR²88

Phase 1 : réduction de la taille du problème par élimination de contraintes et de variables .

Dans ce chapitre , nous allons exposer l'algorithme général de la nouvelle mise en œuvre de PR² . Ce nouvel algorithme tient compte des remarques du chapitre précédent .

Nous allons décrire l'algorithme que nous nous proposons de mettre en œuvre .

Algorithme

Début
Réductions évidentes par R1,R2,R3 ;
Calcul d'une suite de W par l'algorithme de sous-gradient ;
Sélection d'une sous-suite de W ;
Lancer les tâches "traitement parallèle" ;
Exécuter "traitement parallèle" ;
Attendre la fin d'exécution des tâches "traitement parallèle" ;
Fin

Tâches traitement parallèle

Paramètre transmis : k , le numéro de la tâche
La variable X1 contient l'état des variables (fixées ou non) et est partagée par les tâches . La variable tx joue le même rôle que X1 mais est locale à chaque tâche .

```
Début

    Initialiser les variables locales ;

    Tant que non (fin phase 1) et non (fin tâche k) faire

        Recopier X1 dans tx ;
        Tant que non (toutes les variables sont fixées ou toutes les
        contraintes sont traitées ou tous les w_i sont traités) Faire
        |   Prendre une contrainte i ou un multiplicateur wi en section
        |   critique ;
        |   Procédure de fixation des variables ;
        |   Mettre à jour tx en fonction des variables fixées ;
        |   Si le nombre de variables éliminées depuis la dernière mise à
        |   jour  est assez grand
        |   |   alors
        |   |       Mettre à jour X1 et MAJ en section critique ;
        |   |   sinon
        |   |       Si une autre tâche a effectué une mise à jour alors
        |   |       |   Recopier X1 dans tx en section critique
        |   |       Fsi ;
        |   Fsi ;
        Fait ;

        Rendez-vous avec les autres tâches ;

        Attendre que toutes les tâches soient au rendez-vous ;

        Si la tâche k est la dernière tâche arrivant au rendez-vous alors
            (toutes les autres tâches sont en attente)
            Mettre à jour le problème réduit en fonction des résultats de
            chacune des tâches ;
            Si le problème n'est pas entièrement résolu alors
                Si des variables ont été éliminées alors
                    Copier X1 dans tous les tx (pour chaque tâche) ;
                    Calculer une nouvelle suite W ;
                    Sélectionner une nouvelle sous-suite ;
```

```
                    Sinon
                        Fin de la phase 1
                    Fsi ;
                Sinon
                    Fin de la réduction ;
            Fsi ;
            Réveiller les autres tâches ;
        Fsi ;
    Fait ;

    Elimination des contraintes

    Si non problème résolu et non fin de la tâche k alors
        Calculer une nouvelle suite W ;
        L'ensemble des multiplicateurs composites devient :
                { w* , e1 , e2 , ... , em }
        Tant que toutes les contraintes ne sont pas traitées faire
            Prendre une contrainte à traiter en section critique ;
            Tant que il reste des contraintes j non éliminées et non
            utilisées ou des wj non utilisés Faire
                Elimination de la contrainte i avec B^i_j ou B^i(w*) ;
            Fait ;
        Fait ;
    Fsi ;

Fin
```

CHAPITRE 5 : ANALYSE DES RESULTATS DE PR² 88

réduction de la taille

Nous avons testé notre programme sur 50 problèmes tests de la littérature . Ce programme a été implémenté sur le CRAY 2, en fortran 77 (environ 3500 lignes de code) . Les tableaux des pages suivantes contiennent les résultats des réductions après exécution du programme séquentiel FPR 83 , et des deux programmes parallèles $PR^2$87 et PR^2 88 .

Nous remarquons que notre algorithme fournit toujours des résultats , en termes de nombre de variables fixées , au moins aussi bons que ceux des deux autres programmes . Ceci s'explique par le fait que nous mettons en œuvre un nombre plus important de tests . En outre , nous obtenons de **meilleurs résultats pour 27 problèmes** . Il est à noter, que pour le deuxième problème de Senju et Toyoda (ST2) , nous obtenons un très bon résultat , puisque nous fixons 18 variables de plus que PR^2 87 et 4 variables de plus que FPR 83 , et que nous éliminons 4 contraintes supplémentaires .

En revanche , les temps d'execution de PR^2 88 sont plus élevés que ceux de PR^2 87 , mais demeurent très faibles . Nous pouvons prévoir , que pour les problèmes de plus petites tailles , nous obtiendrons des temps de l'ordre de quelques centaines de millisecondes , de quelques secondes pour les problèmes tels que ceux de Senju et Toyoda (ST1 et ST2) . De plus nous avons testé ce programme avec un problème de plus grande taille : 10 contraintes et 200 variables . Nous avons réduit sa taille à 6 contraintes et 181 variables , ceci en environ 2 minutes , en séquentiel , et environ 1 minute en parallèle (avec 4 tâches) . Ceci laisse supposer que nous avons un speed-up intéressant . Des tests ultérieurs devront confirmer ces premiers résultats , plus particulièrement , en ce qui concerne des problèmes de plus grande taille .

Tableau des résultats

Problème : nom du problème
Taille : taille du problème d'origine : nombre de contraintes x nombre de variables
FPR 83 : taille du problème réduit après exécution du programme séquentiel FPR 83
PR^2 87 : taille du problème réduit après exécution du programme parallèle PR^2 87
PR^2 88 : taille du problème réduit après exécution du programme parallèle PR^2 88
temps $PR^2$87 : temps d'exécution pour le programme PR^2 87

Il est à noter que notre programme résout (par énumération explicite) les problèmes dont la taille réduite contient moins de 10 variables . Ces problèmes peuvent donc être considérés comme étant résolus (ces problèmes sont signalés par *)

Problème	Taille	FPR 83	PR^2 87	PR^2 88	temps PR^2 87
Pet 1	10 x 6	0 x 0	0 x 0	0 x 0	0,018
Pet 2	10 x 10	1 x 3	1 x 3	1 x 3 *	0,024
Pet 3	10 x 15	5 x 8	5 x 8	5 x 8 *	0,038
Pet 4	10 x 20	2 x 8	2 x 10	2 x **7** *	0,043
Pet 5	10 x 28	2 x 9	2 x 10	**0 x 0**	0,055
Pet 6	5 x 39	4 x 28	5 x 38	4 x **27**	0,045
Pet 7	5 x 50	4 x 38	5 x 46	4 x **36**	0,068
NW 1	2 x 28	1 x 4	1 x 4	1 x 4 *	0,031
NW 2	2 x 28	1 x 5	2 x 11	**0 x 0**	0,028
NW 3	2 x 28	2 x 19	2 x 21	2 x**14**	0,025
NW 4	2 x 28	1 x 5	1 x 5	**0 x 0**	0,031
NW 5	2 x 28	0 x 0	0 x 0	0 x 0	0,031
NW 6	2 x 28	2 x 7	2 x 6	2 x **5** *	0,029
NW 7	2 x 105	2 x 12	2 x 61	2 x 12	0,106
NW 8	2 x 105	2 x 29	2 x 102	2 x **28**	0,070
HP 1	4 x 28	4 x 27	4 x 28	4 x **26**	0,023
HP 2	4 x 35	4 x 34	4 x 34	4 x 34	0,033
ST 1	30 x 60	30 x 40	30 x 53	30 x 40	0,574
ST 2	30 x 60	30 x 37	30 x 51	**26 x 33**	0,556

Problème	Taille	FPR 83	PR^2 87	PR^2 88
WS 1	5 x 30	5 x 14	5 x 21	5 x **12**
WS 2	5 x 30	1 x 5	4 x 13	**0 x 0**
WS 3	5 x 30	4 x 11	4 x 13	**4 x 7** *
WS 4	5 x 30	1 x 2	2 x 5	**0 x 0**
WS 5	5 x 30	0 x 0	0 x 0	0 x 0
WS 6	5 x 40	3 x 11	5 x 23	**2 x10***
WS 7	5 x 40	3 x 11		3 x **8** *
WS 8	5 x 40	2 x 10		2 x **5** *
WS 9	5 x 40	1 x 3	0 x 0	
WS 10	5 x 50	3 x 18		3 x **17**
WS 11	5 x 50	2 x 12		2 x 12
WS 12	5 x 50	1 x 5		
WS 13	5 x 50	3 x 15		**0 x 0**
WS 14	5 x 60	3 x 11		3 x 11
WS 15	5 x 60	3 x 6		
WS 16	5 x 60	2 x 8	3 x 10	
WS 17	5 x 60	1 x 14	1 x 9	**0 x 0**
WS 18	5 x 70	3 x 13	3 x 27	3 x **11**
WS 19	5 x 70	1 x 2	3 x 18	
WS 20	5 x 70	3 x 9	5 x 19	3 x 9 *
WS 21	5 x 70	1 x 7	1 x 7	1 x **5** *
WS 22	5 x 80	3 x 29	3 x 20	3 x **10***
WS 23	5 x 80	2 x 13	3 x 20	
WS 24	5 x 80	3 x 20	3 x 13	
WS 25	5 x 80	3 x 10	4 x 18	3 x **9** *
WS 26	5 x 90	3 x 30	3 x 22	**2 x 8** *
WS 27	5 x 90	2 x 7	3 x 12	2 x **6** *
WS 28	5 x 90	2 x 8	3 x 14	**0 x 0**
WS 29	5 x 90	2 x 9	2 x 13	**0 x 0**
WS 30	5 x 90	0 x 0	0 x 0	0 x 0

Nous allons détailler les résultats obtenus afin de pouvoir déterminer l'incidence de l'utilisation des multiplicateurs composites (et des nouveaux knapsacks outils) et l'apport du test dît des "relations binaires" . les deux tableaux de la page suivante illustrent ces résultats .

Nous remarquons que le multiplicateur composite considéré comme étant optimal w^* réalise la plus grande partie des fixations des variables . Ceci explique en partie que nous obtenons de meilleurs résultats que PR^2 88 . D'autre part , le test dît des "relations binaires" permet de fixer certaines variables supplémentaires .

Détail des résultats obtenus

Taille : taille du problème initial contraintes x variables
Total : nombre de variables fixées lors de la phase de réduction
W* 1 : nombre de variables fixées par le multiplicateur composite optimal w* lors de la première itération de la phase de fixation des variables
W* 2 : nombre de variables fixées par le multiplicateur composite optimal w* lors de la deuxième itération de la phase de fixation des variables
Essai : nombre de variables fixées par le test dît des "relations binaires"
Indice : numéro du rang du multiplicateur optimal (de la permière itération) w* dans la sous-suite engendrée par l'algorithme du sous-gradient (au maximum 40)

Problème	Taille	TOTAL	W* 1	W* 2	ESSAI	indice
Pet 1	10 x 6	6	3	0	0	10
Pet 2	10 x 10	7	5	0	0	4
Pet 3	10 x 15	7	4	0	0	18
Pet 4	10 x 20	13	11	0	1	12
Pet 5	10 x 28	28	17	0	2	16
Pet 6	5 x 39					
Pet 7	5 x 50					
NW 1	2 x 28	24	17	0	0	4
NW 2	2 x 28	28	17	0	1	6
NW 3	2 x 28	14	8	0	6	4
NW 4	2 x 28					
NW 5	2 x 28	28	24	0	0	2
NW 6	2 x 28	23	22	0	1	2
NW 7	2 x 105	90	40	0	0	10
NW 8	2 x 105	77	64	0	10	2
HP 1	4 x 28					
HP 2	4 x 35					
ST 1	30 x 60					
ST 2	30 x 60					

Problème	Taille	TOTAL	W* 1	W* 2	ESSAI	indice
WS 1	5 x 30	18	17	0	1	16
WS 2	5 x 30	30	19	2	0	14
WS 3	5 x 30	23	19	0	2	12
WS 4	5 x 30	30	26	0	1	6
WS 5	5 x 30	30	26	0	0	10
WS 6	5 x 40	30	27	0	1	10
WS 7	5 x 40	32	28	1	2	22
WS 8	5 x 40	35	30	0	0	12
WS 9	5 x 40					
WS 10	5 x 50	33	34	0	2	30
WS 11	5 x 50	38	38	0	1	14
WS 12	5 x 50					
WS 13	5 x 50	50	32	3	3	18
WS 14	5 x 60	48	36	6	2	36
WS 15	5 x 60					
WS 16	5 x 60					
WS 17	5 x 60	60	51	0	1	34
WS 18	5 x 70	59	54	2	2	24
WS 19	5 x 70	57	40	1	4	16
WS 20	5 x 70	61	60	0	0	10
WS 21	5 x 70	65	56	0	1	8
WS 22	5 x 80	70	54	2	0	34
WS 23	5 x 80					
WS 24	5 x 80					
WS 25	5 x 80	71	67	0	2	12
WS 26	5 x 90	82	51	4	7	20
WS 27	5 x 90	84	74	0	9	22
WS 28	5 x 90	90	57	4	5	20
WS 29	5 x 90	90	56	0	9	24
WS 30	5 x 90					

CHAPITRE 6 : LA RESOLUTION EXACTE DU PROBLEME DU MULTIKNAPSACK EN PARALLELE

1. INTRODUCTION

Après avoir appliqué l'algorithme de réduction à un problème de multiknapsack , nous disposons d'un problème de taille réduite que nous devons résoudre .

Soient X_1 et X_0 les ensembles des variables fixées respectivement à 1 et à 0 au cours de la phase de réduction et X_2 l'ensemble des variables non fixées . Le problème réduit peut s'écrire de la manière suivante :

$$\begin{aligned} \text{Max} \quad & \sum_{j \in X_2} c_j \cdot x_j \\ \text{s.c.} \quad & \sum_{j \in X_2} a_{l,j} \cdot x_j \leq b_l \quad , \; l \text{ contrainte non éliminée} \\ & x_j \in \{0,1\} \end{aligned}$$

Afin de simplifier cet exposé , nous supposerons par la suite que nous travaillons sur le problème d'origine . Les problèmes traités seront notés de la manière suivante :

$$\begin{aligned} (B) \quad \text{Max} \quad & c \cdot x \\ \text{s.c.} \quad & A \cdot x \leq b \\ & x \in \{0,1\}^n \end{aligned}$$

Il s'agit d'un problème à n variables et m contraintes .

Afin de résoudre ce problème , nous avons choisi d'utiliser une méthode de résolution implicite , dîte par Séparation et Evaluation (ou Branch and Bound) . Le principe général de ces méthodes est présenté en annexe .

Nous rappelons que ces méthodes se différencient suivant trois paramètres :

- la fonction d'évaluation
- le principe de séparation
- la stratégie de parcours de l'arborescence

2 :PRINCIPE GENERAL D'UN ALGORITHME PARALLELE POUR LES METHODES DE RESOLUTION PAR SEPARATION ET EVALUATION

Nous allons présenter un algorithme général parallèle de résolution par une méthode par séparation et évaluation ([ROU]) .

UN ALGORITHME B&B PARALLELE

Nous allons présenter un algorithme B&B parallèle . Cet algorithme est particulièrement bien adapté aux machines de type CRAY 2 dont le nombre de processus est en général réduit et les délais de communication entre les processeurs et la mémoire très rapides . L'algorithme se compose de tâches identiques , effectuant toutes le même traitement sur différents sommets de l'arborescence . Seule une tâche privilégiée , la tâche initiale sera chargée des initialisations et du lancement des autres tâches . La répartition du travail entre les tâches est réalisée en donnant accès à une liste partagée contenant tous les sommets à séparer ainsi que les informations les définissant (ce que nous appelons le contexte du sommet et qui dépend du problème traité) .

Trois types d'opérations peuvent être effectuées sur cette liste .

a) Sélection d'un élément

L'orsqu'une tâche est inactive , suivant la stratégie de construction de l'arborescence définie , elle choisit un sommet de la liste .

b) Insertion d'un élément

Lorsque la sélection d'un sommet engendre plusieurs successeurs dont l'évaluation est supérieure à la meilleure solution déjà trouvée , ces successeurs sont insérés dans la liste .

c) Suppression d'un élément

Elle se fait dans deux cas . Lorsqu'une tâche libre prend un sommet dans la liste (a) , moins trivialement lorsque la valeur de la meilleure solution connue est améliorée . Tous les sommets d'évaluation inférieure à cette valeur sont éliminés puisqu'ils ne peuvent plus conduire à une solution meilleure .

La valeur de la meilleure solution connue est une vartiable partagée remise à jour par les tâches , qui trouvent lorsqu'elles séparent un sommet une solution réalisable locale de meilleure valeur .

La terminaison de l'algorithme est détectée lorsque la liste est vide et toutes les tâches inactives (en utilisant les primitives de synchronisation) .

2. Variables employées

Nous donnons ici les définitions des variables nécessaires à la compréhension de l'algorithme .

- Variables partagées

LISTE liste dont chaque élément contient l'évaluation d'un sommet et les informations nécéssaires à la séparation .
NBLIST nombre de sommets présents dans la liste .
BI meilleure borne inférieure déjà trouvée .
NTA nombre de tâches actives .
FIN arrêt de la tâche .

- Evènement et sémaphore

INSER evènement envoyé aux tâches en attente lorsque'une tâche insère des sommets dans la liste alors que celle-ci est vide , ou lorsque l'algorithme est fini et qu'il faut réveiller toutes les tâches en attente afin d'avoir une terminaison correcte .
MUTEX sémaphore d'exclusion mutuelle .

- Variables locales à une tâche .

binf borne inférieure locale associé au sommet nséparé .
bsup borne supérieure associée à chaque sommet créé par séparation .

3. Algorithme

Tâche initiale

```
NTA := 0 ;
NBLIST := 0 ;
FIN := faux ;
Séparation de la racine ;
Evaluation de ses successeurs (bsup) ;
Calcul d'une borne inférieure (binf) ;
BI := binf ;
Pour chaque sommet successeur faire
  si (bsup > BI) alors
     insérer le successeur dans LISTE ;
     NBLIST := NBLIST + 1 ;
  Fsi ;
Fait ;
Lancer les tâches "traitement parallèle" ;
Exécuter "traitement parallèle" ;
```

```
Tâche "traitement parallèle

début
 FIN := faux ;
 Tant que ( non(FIN) ) faire
   Verouiller MUTEX ;
   Si ( NBLIST = 0 ) alors
        Si ( NTA = 0 ) alors
             FIN := vrai ;
             Dévérouiller (MUTEX) ;
             Envoyer évènement (INSER) ;
          sinon
             Dévérouiller (MUTEX) ;
             Attendre évènement (INSER) ;
        Fsi ;
      sinon
        NTA := NTA + 1 ;
        Sélectionner un sommet dans la liste LISTE ;
        Supprimer cet élément de LISTE ;
        NBLIST := NBLIST - 1 ;
        Dévérouiller (MUTEX) ;
        Séparation : créer les successeurs du sommet sélectionné ;
        Calcul de la borne inférieure binf ;
        Calcul de la borne supérieure de chaque successeur (bsup);
        Si (binf > BI) alors
          Vérouiller (MUTEX)
          Supprimer de LISTE tous les sommets tels que bsup ≤ BI ;
          Décrémenter NBLIST d'autant ;
          BI := binf ;
          Dévérouiller (MUTEX) ;
        Fsi ;
        Vérouiller (MUTEX) ;
        Pour chaque sommet successeur faire
          Si (bsup > BI) alors
            Si (NBLIST = 0) alors
              envoyer évènement (INSER) ;
            Fsi ;
            Insérer ce successeur dans LISTE ; NBLIST := NBLIST + 1 ;
          Fsi
        Fait
        NTA := NTA - 1 ;
        Dévérouiller (MUTEX)
      Fsi ;
  Fait
Fin
```

3 : UN NOUVEL ALGORITHME PARALLELE DE RESOLUTION EXACTE DU PROBLEME DU MULTIKNAPSACK

I- PRINCIPE GENERAL

Cet algorithme utilise les notions de relaxations et de variables de bases liées à un knapsack , qui sont expliquées en annexe () .

Nous allons présenter la fonction d'évaluation et le principe de séparation mis en œuvre par cet algorithme .

a) Notations

Soit E_k un nœud de l'arborescence , associé à un sous-ensemble de solutions S_k du problème (B) .
L'ensemble S_k contient toutes les solutions admissibles d'un sous-problème (B_k) de (B) . Le problème (B_k) est un multiknapsack déduit de (B) par la fixation de variables .
Soient les ensembles de variables X_{k0} et X_{k1} les ensembles des variables fixées respectivement à 0 et à 1 dans le problème (B_k) :

$$(B_k) = (B \mid \forall x \in X_{k0} , x = 0 \ ; \quad \forall x \in X_{k1} , x = 1)$$

Notons n_k le nombre de variables actives du problème (B_k) et m_k le nombre de contraintes non éliminées (par le test trivial R2) .

Par la suite chaque nœud E_k sera associé au problème (B_k) plutôt qu'au sous-ensemble de solutions S_k .

b) La fonction d'évaluation

Soit un nœud E_k associé au problème (B_k) .
Soit la fonction d'évaluation ev , qui à chaque nœud associe son évaluation. Cette fonction doit vérifier les propriétés suivantes :

(1) $ev(E_k) \geq Max_{s \in S_k} \ v(s)$

où v(s) est la valeur de la solution réalisable s
donc , $ev(E_k) \geq v(B_k)$

(2) Soient E_{k1} , E_{k2} , ... , E_{kp} les successeurs de E_k

$$ev(E_k) \geq ev(E_{kj}) \ , \ \forall j \in \{1,p\}$$

Lors de la réduction , nous avons introduit la notion de "meilleur multiplicateur composite" . Nous allons utiliser ce multiplicateur afin d'obtenir de "bonnes" évaluations du problème associé à chaque nœud. Notre fonction d'évaluation peut donc s'écrire de la manière suivante :

Soit un noœud E_k $\quad ev(E_k) = v(\overline{B_k(w^*)})$

où w^* est le multiplicateur composite optimal associé au problème (B_k)

et $(B_k(w^*))$ la relaxation composite associée au problème (B_k) .

$$B_k(w^*) \quad \text{Max } c.x$$
$$\text{s.c. } w^*Ax \leq w^*b$$
$$x \in \{0,1\}^n$$

Nous pouvons donc espérer obtenir de bons majorants de $v(B_k)$.

D'autre part , afin d'obtenir plus rapidement des problèmes de petite taille , nous avons décider d'appliquer quelques tests de réduction à chacun des sous-problème traité . Nous appliquons à chaque problème les tests de réduction triviale , et le test V1 (relaxation Lagrangienne) associé à tous les multiplicateurs Lagrangiens sur le knapsack outil $(B_k(w^*))$.

c) Le principe de séparation

Cet algorithme repose sur un principe de séparation dichotomique , c'est-à-dire que chaque nœud peut donner naissance à deux autres nœuds .

Soit un nœud E_k

Soit i* la variable de base de ce problème $(\overline{B_k(w^*)})$

Le nœud E_k donne naissance à deux nœuds E_{k0} et E_{k1} associés respectivement aux problèmes (B_{k0}) et (B_{k1}) .

$$(B_{k0}) = (B_k \mid x_{i^*} = 0) \quad \text{et} \quad (B_{k1}) = (B_k \mid x_{i^*} = 1)$$

La fixation d'une variable permet de diminuer la taille des problèmes , et donc fournit des problèmes plus faciles . D'autre part nous pouvons vérifier que la fonction d'évaluation vérifie la propriété (2) .

c) Traitement effectué en chaque nœud

Soit E_k un nœud de l'arborescence . Nous allons détailler le traitement effectué lors de sa séparation et de l'évaluation de ses fils . Ces traitements sont enchaînés de la manière suivante :

- Tests de réduction triviale appliqués à (B_k)
- Calcul du multiplicateur optimal w^* associé au problème (B_k) réduit
- Test de réduction V_1 appliqué à $(B_k(w^*))$
- Détermination de la variable de base i^* de $(B_k(w^*))$
- Détermination des évaluations des problèmes $(B_k(w^*)|x_{i^*}=0)$ et $(B_k(w^*)|x_{i^*}=1)$ (évaluations des nœuds fils)

Les deux premiers points ont déjà été traités lors de la phase de réduction . Nous allons nous intéresser plus particulièrement aux liens qui existent entre les autres traitements .

Le test de réduction V1

Lors du test V1 , nous avons vu que nous sommes amenés à calculer deux matrices , la matrice CR des coûts réduits et la matrice V des valeurs des relaxations . A l'issu du test , un certain nombre de variables a pu être fixé. De même que pour le test des relations binaires mis en place lors de la réduction , nous modifions ces deux matrices en fonction des variables fixées :

- suppression des lignes et des colonnes correspondant aux variables fixées
- modification des valeurs des relaxations (matrice V)

Détermination de la variable de base

De plus , nous savons que la variable de base i^* correspond à la meilleure relaxation Lagrangienne , c'est-à-dire que :

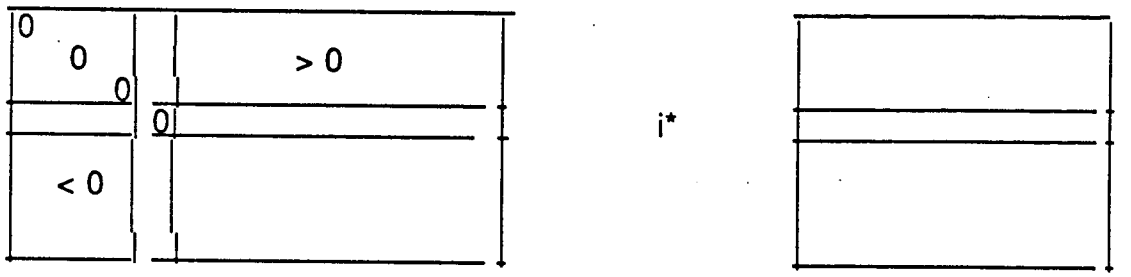
$$V(i^*) = \min_{i=1,\ldots,nk} V(i)$$

Il est donc facile de déterminer l'indice de la variable de base .

Exemple

Nous disposons des matrices V et CR . Pour simplifier , supposons que les variables sont triées de telle sorte que $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. ($\lambda_i = c_i / w^*A^i$)

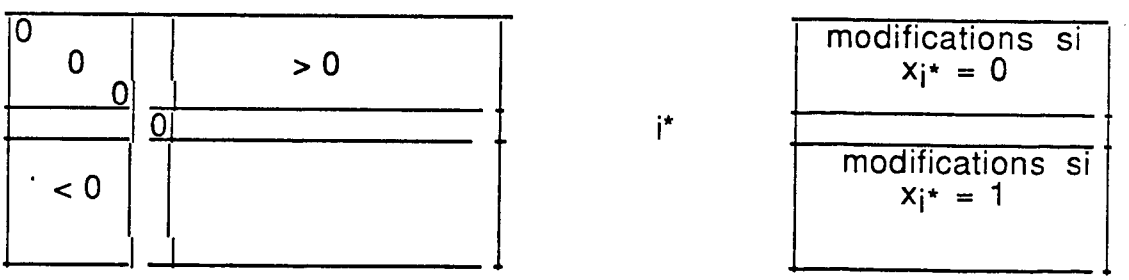
Nous pouvons montrer que les matrices CR et V se présentent de la manière suivante :



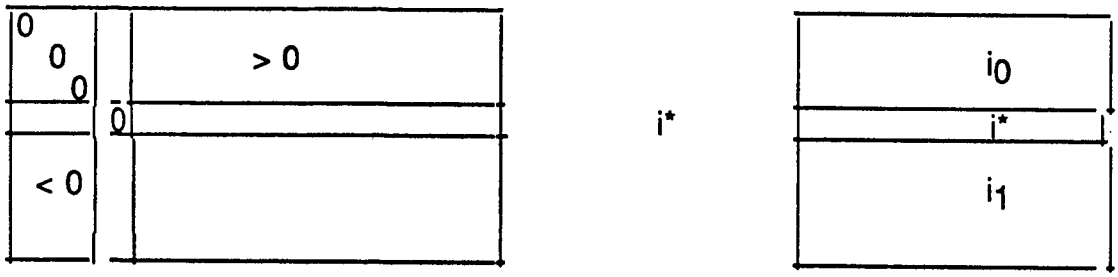
CR V

Les coûts réduits situés au-desus de la diagonale sont positifs , tandis que ceux qui sont situés au-dessous sont négatifs . La variable x_{i^*} est déterminée par le fait qu'elle minimise les valeurs de la matrice V .

Considérons le problème dans lequel x_{i^*} est fixée à 0 (respectivement à 1) . Dans ce cas , les valeurs des relaxations du haut (au-dessus de la lignes i*) (resp. du bas , au-dessous de la ligne i*) seront donc modifiées puisque les coûts réduits associés à ces lignes et à la colonne i* (cr(i,i*)) sont positifs (resp. négatifs) .



En conséquences la ligne de l'indice i_0 (resp i_1) de la variable de base du problème lorsque x_{i^*} est fixée à 0 (resp. à 1) se trouve au-dessus (resp. au-dessous) de la ligne i^* , et correspond à la ligne ayant la valeur minimale des lignes se situant au-dessus (resp. au-dessous) de la ligne i^* (après modification) .



Nous rappelons que V(i*) est égale à la valeur de la relaxation Lagrangienne du problème $(B_k(w^*))$, compte tenu des fixations précédentes , de laquelle a été soustrait la valeur du minorant de $(B_k(w^*))$ (déterminé par l'heuristique au début)
(pour simplifier les notations , par la suite , nous noterons $(B_k(w^*)')$ le problème $(B_k(w^*))$ réduit)

$$V(i^*) = v(RL(\lambda_{i^*}, B_k(w^*)')) - \underline{v}(B_k')$$

Séparation du nœud

Nous devons calculer les valeurs des problèmes $\overline{(B_k(w^*)'|x_{i^*}=0)}$ et $\overline{(B_k(w^*)'|x_{i^*}=0)}$

Soit ε égal à 0 ou 1 . Calculons la valeur de $\overline{(B_k(w^*)' \mid x_{i^*} = \varepsilon)}$
Pour cela , nous mettons à jour la matrice V en fonction de cette fixation :

Soit $i \in \{1,\ldots,n_k\}$ (x_i n'a pas été fixée précédemment)
V(i) = $\lambda_i w^* b$ + somme des coûts réduits cr(i,j) positifs - $\underline{v}(B_k')$

Par exemple , si nous fixons x_{i^*} à 0 , et si le coût réduit associé cr(i,i*) est positif , nous devons soustraire ce coût réduit de V(i) , puisque dans la relaxation i nous avons considéré que x_{i^*} valait 1 .

Donc $\forall i \in \{1,\ldots,n_k\}$
si ε=0 et cr(i,i*) > 0 alors V(i) := V(i) - |cr(i,i*)|
si ε=1 et cr(i,i*) < 0 alors V(i) := V(i) - |cr(i,i*)|

Nous avons ainsi les valeurs des relaxations Lagrangiennes après fixation de x_{i^*} à ε . La valeur du problème $(B_k'|x_{i^*}= \varepsilon)$ est égale au minimum des valeurs de ces relaxations .
$v(\overline{B_k'|x_{i^*}= \varepsilon}) = \min_i V(i)$, V(i) ayant été modifié en fonction de la fixation de x_{i^*} .

En résumé $v(\overline{B_k'|x_{i^*}= 0}) = \min_i V(i)$ - cr(i,i*) positifs
$v(\overline{B_k'|x_{i^*}= 1}) = \min_i V(i)$ + cr(i,i*) négatifs
Il est donc facile de déterminer ces deux valeurs à partir des matrices CR et V .

Algorithme

Si le problème n'est pas réalisable (aucune solution réalisable ne peut être trouvée) alors
  Fin du traitement du nœud ;
Fsi ;

Application des tests de réduction triviaux à (B_k) ;
Si il reste plus de 10 variables actives alors

Calcul du multiplicateur optimal w* (sous-gradient) (le multiplicateur initial est le multiplicateur optimal associé au nœud père de E_k);
Calcul de la matrice des coûts réduits ;
Calcul de la matrice V ;
Application du test V1 ;

Si il reste plus de 10 variables actives alors
  Mise à jour des matrice V et CR :

Pour toutes les variables x_j fixées par V1 Faire
| Pour toutes les lignes i de la matrice V Faire
| | Si (x_j est fixée à 0 et cr(i,j)>0) alors V(i) = V(i) - cr(i,j) Fsi ;
| | Si (x_j est fixée à 1 et cr(i,j)<0) alors V(i) = V(i) + cr(i,j) Fsi ;
| Fait ;
| Supprimer la colonne j et la ligne j de CR ;
| Supprimer la ligne j de V ;
Fait ;

Détermination de la variable de base i* :
  $V(i^*) = \min_i V(i)$
Détermination de i_0 et i_1 tels que :
  $V(i_0) = \min_i$ (V(i) - cr(i,i) positifs)
  $V(i_1) = \min_i$ (V(i) - cr(i,i) négatifs)
  $i\varepsilon$ est la variable de base du problème $(B_k'|x_{i^*}= \varepsilon)$
  $v(B_k'|x_{i^*}= \varepsilon) = V(i\varepsilon)$;
  Séparation du nœud en deux nœuds associés aux problèmes $(B_k'|x_{i^*}= 0)$ et $(B_k'|x_{i^*}= 1)$;

Si $V(i_0)$ > BI alors
insertion du nœud dans l'arbre ;
Fsi ;
Si $V(i_1)$ > BI alors
insertion du nœud dans l'arbre ;
Fsi ;

sinon

Résolution explicite du problème (par énumération de toutes des solutions realisables) ;
Si la valeur de la solution > BI alors
La nouvelle solution devient la meilleure solution ;
Elimination des nœuds de moins bonne évaluation ;
Fsi ;
Fsi ;

sinon

Résolution explicite du problème (par énumération de toutes des solutions realisables) ;
Si la valeur de la solution > BI alors
La nouvelle solution devient la meilleure solution ;
Elimination des nœuds de moins bonne évaluation ;
Fsi ;
Fsi

Il est à noter que le traitement d'un nœud ne fait pas intervenir l'algorithme NKR de résolution des knapsacks en continu . En effet tous les traitements peuvent être effectués à partir de la seule donnée des matrices des coûts réduits et des valeurs des relaxations .

II Traitement du parallèlisme

Nous pouvons espérer que l'introduction du parallèlisme va nous permettre d'obtenir des temps de résolution acceptables des problèmes de multiknapsacks . En effet , nous allons pouvoir traiter plusieurs nœuds simultanément , ce qui nous permettra d'atteindre des solutions réalisables plus rapidement , donc de meilleures bornes inférieures , et evidemment de parcourir l'arborescence plus rapidement .

Un algorithme parallèle mettant en œuvre des méthodes par Séparation et Evaluation est proposé en annexe . Toutes les tâches effectuent le même traitement , c'est-à-dire le traitement des nœuds actifs . De plus toutes les tâches sont chargées de la gestion de l'arborescence (insertions de nœuds , suppressions , ...) , ce qui nous a conduit à introduire un certain nombre de sections critiques , donc de sémaphores et d'évènements . D'autre part , une tâche principale effectue les initialisations , crée le nœud racine et ses deux fils , puis crée les autres tâches .

Nous allons donner un algorithme général puis nous étudierons les problèmes que génèrent le parallèlisme .

Tâche principale

```
Début
  Initialisations générales ;
  Séparation de la racine ;
  Création et insertion dans l'arbre des deux nœuds fils ;
  Lancement des tâches "traitement parallèle" ;
  Exécution du traitement parallèle ;
  Attente des autres tâches ;
Fin
```

Tâches "traitement parallèle

```
Début
  Demander un nœud actif à traiter ;
  Si un nœud actif à traiter alors
  |    Traiter le nœud et Insérer les nœuds fils dans l'arborescence ;
  |  sinon
  |    Si toutes les autres tâches attendent un nœud actif alors
  |    |    Fin du programme ;
  |    |    Indiquer aux autres tâches que le programme est terminé ;
  |    |  sinon
  |    |    Attendre qu'un nœud actif soit inséré dans l'arbre ;
  |    Fsi ;
  Fsi ;
Fin ;
```

Un certain nombre de problèmes apparaissent , plus particulièrement sur deux points : la cohérence de l'information du fait du partage des données et la gestion de la terminaison des tâches .

Nous disposons d'une structure de données contenant les nœuds actifs de l'arborescence . Etant donné que nous utilisons le Fortran , il n'a pas été possible d'utiliser une structure dynamique . Nous avons choisi d'utiliser un tableau (ou plutôt plusieurs tableaux) , chaque ligne de ce tableau représentant un nœud actif . Nous détaillerons la structure de ce tableau par la suite .

Les principaux problèmes concernent :

- La gestion du tableau
- La gestion des nœuds actifs de l'arborescence
- Le contrôle de la granularité (répartition du travail entre les tâches)
- La terminaison des tâches

Nous allons détailler chacun de ces points .

1- La gestion du tableau

Le tableau est une donnée partagée . Les problèmes interviennent au niveau de l'allocation d'une ligne (ou place) libre à une tâche voulant insérer un nœud dans l'arborescence , et de la gestion des places libres . Pour cela nous utilisons un tableau logique LIBRE indiquant si une place est libre : LIBRE(i) est vrai si la ligne i est libre .

Pour allouer une place , il suffit de trouver la première place dont la valeur dans LIBRE est vraie . Mais un système de gestion particulier doit être mis en place lorsqu'il ne reste plus aucune place libre . Dans ce cas deux cas se présentent :

- D'autres tâches sont actives et vont peut-être libérer des places. Dans ce cas la tâche demandant une place est mise en attente , jusqu'à ce qu'une place soit libre ou jusqu'à la fin du programme

- Toutes les autres tâches sont en attente . Dans ce cas , aucune place ne pourra être libérée , nous décidons que le programme est terminé . La tâche demandant une place réveille les autres tâches et leur indique que le programme est terminé .

Ceci nous a conduit à introduire des sémaphores et des évènements :

- **Sémaplace** Sémaphore contrôlant l'accès à la structure des données associée à l'arbre .

- **Sémaatpl** Sémaphore protégeant les paramètres utilisés lorsqu'il n'y a pas de place disponible , les tâches demandant une place peuvent alors être mises en attente .

- **Sémafin** Sémaphore permettant la gestion de la terminaison des tâches .

- **Evtf** Evènement envoyé lorsque des tâches attendent la libération d'une place dans l'arborescence et qu'une place est effectivement libérée .

- **Fin** Tableau logique indiquant à chaque tâche si elle a terminé .

Nous allons détailler les algorithmes des procédures de demande et de libération d'une place .

Demande d'une place

```
Début
  Répéter
  | Ouverture du sémaphore Sémaplace ;
  | Recherche d'une place libre ;
  | Si on a trouvé une place libre alors
  |     Allocation de la place à la tâche ;
  |     LIBRE(place) := faux ;
  |     Fermeture du sémaphore semaplace ;
  |   sinon
  |     Fermeture du sémaphore semaplace ;
  |     Si le nombre de tâches actives est égal à 1 alors
  |         Fin du programme ;
  |       sinon
  |         Ouverture du sémaphore semaatpl ;
  |         Si toutes les autres tâches sont en attente alors
  |             Pour toutes les tâches i  fin(i):=vrai ;
  |             Réveiller les tâches en attente sur l'évènement evtf
  |               (attente d'une place) ;
  |             Réveiller les tâches en attente sur l'évènement evtp
  |               (attente d'un nœud actif) ;
  |             Réveiller les tâches en attente sur l'évènement evts
  |               (attente dûe au contrôle de la granularité) ;
  |             Fermeture du sémaphore semaatpl ;
  |             Fin de la tâche ;
  |           sinon
  |             Fermeture du sémaphore semaatpl ;
  |             Attendre l'évènement evtf ;
  |             Ouverture du sémaphore sémafin
  |             Si (fin de la tâche) alors
  |             | Fermeture du sémaphore sémafin ;
  |             | Fin de la tâche ;
  |             Fsi ;
  |             Fermeture du sémaphore sémafin ;
  |         Fsi
  |     Fsi
  | Fsi
  Jusqu'à on a trouvé une place libre ;
Fin
```

Libération d'une place

Une place peut être libérée lorsqu'une tâche prend un nœud pour le traiter ou lorsque des nœuds sont supprimés en conséquence de l'obtention d'une meilleure solution réalisable .

```
Début
| Ouverture du sémaphore sémaplace ;
|   LIBRE(place) := vrai ;
| Fermeture du sémaphore sémaplace ;
| Si il y a plus d'une tâche alors
|   Ouverture du sémaphore semaatpl ;
|   Si il y a des tâche attendant une place libre alors
|   | Réveiller ces tâches en envoyant l'évènement evtf ;
|   Fsi ;
|   Fermeture du sémaphore semaatpl ;
| Fsi ;
Fin
```

2- La gestion des nœuds actifs de l'arborescence

Les nœuds actifs sont gérés par un arbre binaire de recherche (ou heap) . Ils sont triés dans l'ordre décroissant de leur évaluation .

Lorsqu'une tâche demande un nœud à traiter , elle prend le nœud de plus grande évaluation , donc de "tête de liste" .

Toutes les opérations effectuées sur l'arborescence peuvent se ramener à trois traitements de base : l'insertion d'un nœud dans la structure et son élimination , la demande d'un nœud actif par une tâche . De même que pour la gestion du tableau , les principaux problèmes sont rencontrés lorsqu'une tâche demande un nœud actif et qu'il n'y a aucun nœud dans la structure . Nous allons surtout nous intéresser aux problèmes dûs au parallèlisme , et donc au partage des données .

Nous avons introduit les sémaphores et évènements suivants :

- **Sémaarbre** Sémaphore contrôlant l'accès à l'arbre , utilisé lorsqu'une tâche demande un nœud , veut insérer ou détruire un nœud .

- **Sémaatpb** Sémaphore protégeant les paramètres utilisés lorsqu'il n'y a plus de nœuds actifs dans la structure , les tâches demandant un nœud peuvent alors être mises en attente .

- **Sémafin** Sémaphore permettant la gestion de la terminaison des tâches

- **Evtp** Evènement envoyé lorsque des tâche attendent qu'un nœud actif soit disponible et qu'une tâche a effectivement créé des nœuds actifs .

Nous allons détailler les algorithmes d'insertion d'un nœud et d'élimination d'un nœud .

Insertion d'un nœud

```
Début
   Demander une place dans le tableau (avec attente éventuelle) ;
   Ouverture du sémaphore sémafin ;
   Si (non (fin de la tâche)) alors
   |     Fermeture du sémaphore sémafin ;
   |     Ouverture du sémaphore sémaarbre ;
   |       Insérer le nœud à sa place dans l'arbre ;
   |       (s'il s'agit d'une feuille , modification de Bl)
   |     Fermeture du sémaphore sémaarbre ;
   |     Ouverture du sémaphore sémaatpb ;
   |     Si des tâches attendent un nœud actif alors
   |     | Réveiller les tâches en attente en envoyant l'evènement evtp;
   |     Fsi ;
   |     Fermeture du sémaphore semaatpb ;
   |  sinon
   |     Fermeture du sémaphore sémafin ;
   |     Fin de latâche ;
   Fsi ;
Fin
```

Elimination d'un nœud

```
Début
   Libération de la place du nœud éliminé ;
   Ouverture du sémaphore sémaarbre ;
     Elimination du nœud de la structure (modification des pointeurs) ;
   Fermeture du sémaphore sémaarbre ;
Fin ;
```

3- Le contrôle de la granularité

Au cours de l'élaboration de cet algorithme , nous avons rencontré des problèmes de granularité . En effet , certaines tâches traitaient beaucoup plus de nœuds que les autres et les autres tâches passaient beaucoup de temps en attente du fait de l'utilisation des ressources partagées par les tâches travaillant , ce qui diminuait considérablement l'apport du parallèlisme . Afin de pallier cet inconvénient nous avons décidé d'introduire un mécanisme de contrôle de la répartition du travail entre les tâches . Pour cela , lorsqu'une tâche demande un nœud à traiter, et si cette tâche a traité beaucoup plus de nœuds que les autres , nous décidons de faire attendre cette tâche afin que les autres puissent travailler . Il nous faut trouver un compromis pour savoir dans quelle mesure une tâche doit être mise en attente .

Soit nmoyens le nombre moyen de nœuds traité par une tâche (à un moment donné de l'algorithme) . Une tâche i sera mise en attente si le nombre de nœuds qu'elle a traité est dépasse , d'un certain pourcentage , le nombre moyen de nœuds traités par une tâche .

Soit nbfaits(1,j) le nombre de nœuds qu'une tâche a traité .

$$nbmoyens = \sum_{j=1,\ldots,nta} nbfaits(1,j) \;/\; nta$$

$$\text{Si } nbfaits(1,i) > \frac{pourcentage}{100} \times nmoyens$$

alors la tâche i est mise en attente .
(pourcentage est un paramètre entier supérieur à 100)

Pour cela , nous avons introduit les sémaphores et évènements suivants :

- **Sémasync** Sémaphore permettant de la gestion de la synchronisation des tâches dans le cas où l'on a choisi de contrôler la granularité de l'algorithme

- **Evts** Tableau de nta (nombre de tâches) evènements utilisé lors si l'on a décidé de contrôler la granularité . Cet évènement est envoyé à la tâche concernée si cette tâche est en attente et qu'une autre tâche décide de la libérer .

- **Nbfaits** Tableau à deux dimensions . Soit i une tâche , nbfaits(1,i) contient le nombre de nœuds que la tâche a traité , nbfaits(2,i) est logique et est vrai lorsque la tâche i est en attente .

Demande d'un nœud actif

Cette procédure alloue à une tâche le nœud actif de plus grande évaluation . Dans le cas où l'on a choisi de contrôler la répartition du travail , il y a une première phase de contrôle que nous détaillerons dans le paragraphe suivant .

```
Début
  Contrôle de la répartition du travail entre les tâches ;
  Répéter
  |  Ouverture du sémaphore sémaarbre
  |  Si il n'y a pas de nœuds actifs alors
  |  |     Fermeture du sémaphore sémaarbre ;
  |  |     Si le nombre de tâches actives est égal à 1 alors
  |  |     |     Fin du programme ;
  |  |     |   sinon
  |  |     |     Ouverture du sémaphore sémaatpb ;
  |  |     |     Si toutes les autres tâches sont en attente alors
  |  |     |     |     Pour toutes les tâches i  fin(i):=vrai ;
  |  |     |     |     Réveiller les tâches en attente sur l'évènement evtf
  |  |     |     |       (attente d'un place) ;
  |  |     |     |     Réveiller les tâches en attente sur l'évènement evtp
  |  |     |     |       (attente d'un nœud actif) ;
  |  |     |     |     Réveiller les tâches en attente sur l'évènement evts
  |  |     |     |       (attente dûe au contrôle de la granularité) ;
  |  |     |     |     Fermeture du sémaphore semaatpb ;
  |  |     |     |     Fin de la tâche ;
  |  |     |     |   sinon
  |  |     |     |     Fermeture du sémaphore semaatpb ;
  |  |     |     |     Attendre l'évènement evtp ;
  |  |     |     |     Ouverture du sémaphore sémafin
  |  |     |     |     Si (fin de la tâche) alors
  |  |     |     |     |     Fermeture du sémaphore sémafin ;
  |  |     |     |     |     Fin de la tâche ;
  |  |     |     |     Fsi ;
  |  |     |     |     Fermeture du sémaphore sémafin ;
  |  |     |     Fsi
  |  |     Fsi
  |  |   sinon
  |  |     Allocation du problème à la tâche ;
  |  |     Elimination du nœud ;
  |  |     Fermeture du sémaphore sémaarbre ;
  |  Fsi
  Jusqu'à un nœud actif a été alloué ou fin de la tâche ;
Fin
```

Nous allons détailler le traitement effectué lorsqu'une tâche demande un nœud .

La variable pourcentage est nulle lorsqu'on a décidé de na pas contrôler la répartition du travail entre les tâches , sinon elle contient un entier supérieur à 100 .

La variable no-ta contient le numéro de la tâche demandant le problème .

```
Début
  Si il y a plus de 1 tâche  et  pourcentage non nul  alors
  |  Ouverture du sémaphore sémasync ;
  |  Nbfaits(1,no-ta) := nbfaits(1,no-ta) + 1 ;
  |  Calcul de nmoyens ;
  |  Limite = pourcentage/100 * nmoyens ;
  |  Si (nbfaits(1,no-ta) > limite) alors
  |  |     (* Mise en attente de la tâche *)
  |  |     nbfaits(2,no-ta) := vrai ;
  |  |     Fermeture du sémaphore sémasync ;
  |  |     Attente de l'évènement evts(no-ta) ;
  |  |     Ouverture du sémaphore sémafin ;
  |  |     Si (fin de la tâche) alors
  |  |     |  Fermeture du sémaphore sémafin ;
  |  |     |  Fin de la tâche ;
  |  |     Fsi ;
  |  |     Fermeture du sémaphore sémafin ;
  |  |     Ouverture du sémaphore sémasync ;
  |  Fsi ; nbfaits(2,i) := vrai ;
  |  (* Libération éventuelle de tâches en attente *)
  |  Pour toutes les autres tâches i faire
  |  |  Si ( (la tâche est en attente sur evts(i)) et (nbfaits(2,i) < limite))
  |  |  alors
  |  |  |  (* La tâche i peut être libérée puisque le nombre de nœuds *)
  |  |  |  (* qu'elle a traité est devenu inférieur à la limite *)
  |  |  |  Réveiller la tâche i en envoyant l'evènement evts(i) ;
  |  |  Fsi ;
  |  Fait
  Fsi
Fin
```

4- Gestion de la terminaison des tâches

Un problème se pose pour détecter que le programme est terminée . En effet la fin du programme intervient quand :

- Il n'y a plus de nœuds dans l'arborescence et toutes les tâches sont en attente

- Il n'y a plus de place dans le tableau et toutes les tâches sont en attente

Il est impératif que toutes les tâches soient en attente pour décider de la terminaison . En effet , par exemple , s'il n'y a plus de place dans le tableau et une tâche est encore active , il se peut que cette tâche libère de la place , ce qui permettra aux tâches en attente de reprendre le travail . Nous ne décidons donc de la terminaison que lorsqu'une tâche détecte qu'il n'y a plus de place dans le tableau ou plus de nœuds actifs , et que toutes les autres tâches sont en attente .

Les mécanismes de détection de la terminaison ont été détaillée au cours des paragraphes précédents .

V- ANALYSE DES RESULTATS

Nous avons implémenté cet algorithme sur le CRAY 2 , en FORTRAN 77 (environ 2000 lignes de code) . Nous avons testé ce programme sur le problème test de Flesher (10 contraintes et 20 variables) . Ce problème test est considéré comme étant un problème très difficile . En effet , les variables sont presque toutes equivalentes , et il est difficile de décider de la fixation d'une variable à 1 ou à 0 . Nous avons résolu ce problème en environ 6 secondes , en séquentiel , et en 1 à 3 secondes en parallèle avec quatre tâches . Ceci nous permet de prévoir que le parallèlisme est assez efficace .

En outre , nous avons les résultats statistiques suivants :

- l'arborescence critique est de 72 nœuds (c'est-à-dire que 72 nœuds ont été traités avant le nœud père de la solution optimale) .
- 355 nœuds (non terminaux) ont été traités .
- 77 feuilles ont été trouvées (solutions réalisables ou non)
- le tableau a été occupé par 85 nœuds actifs , simultanément , au maximum

Nous pouvons noter que l'arborescence peut contenir au maximum 2^{10} feuilles (puisque nous résolvons les problèmes de moins de 10 variables et que le problème d'origine a 20 variables) , mais que le programme n'a décrit que 77 feuilles , soit 8% . Ceci montre l'efficacité de l'algorithme d'énumération implicite .

D'autres tests ultérieurs devront préciser l'efficacité de notre algorithme, plus particulièrement au niveau des temps d'exécution ,de la granularité et du speed-up .

D'autres variables sont aussi partagées :

- Les pointeurs de gestion de l'arbre
- La valeur de la meilleure solution réalisable trouvée : BI
- Les sémaphores et les évènements

2- Les variables locales à chaque tâche

Sur le CRAY 2 , toutes les variables appartenant à un common sont partagées . Les variables locales peuvent se présenter sous deux formes :

- Les variables n'appartenant pas à un common .

- Dans le cas précédent , lors des appels de procédures ou de fonctions , les variables doivent être passées en paramètres, ce qui implique un certain nombre de copies . Il est parfois préférable d'utiliser des commons . Dans ce cas , il est nécessaire que ces commons ne soient utilisées que par une seule tâche . Prenons par exemple une variable locale var . Nous la déclarons dans un common sous la forme d'un tableau de variables (la cardinalité du tableau est égale au nombre de tâches) var(nombre de tâches) . A l'intérieur d'une tâche , nous accédons à la valeur locale de la variable par var(numéro de la tâche) .

Chaque tâche utilise les données suivantes :

- Les caractéristiques du nœud qu'elle traite . Celles-ci sont recopiées dans des variables locales dès que le nœud a été alloué à la tâche , afin de limiter le temps d'accès à l'arbre , donc le temps des sections critiques . Ces données sont stockées dans des commons du type précédent .

- Les données locales utilisées pour les calculs : les données du problème associé au nœud (dans un common) , les matrices CR et V (en variables locales) .

III- Structure des données

Dans ce paragraphe , nous allons présenter les structures de données utilisées , d'une part la structure des données partagées (les matrices des données , et le tableau contenant les nœuds actifs) , et d'autre part les données locales utilisées par chaque tâche pour le traitement de chaque nœud .

1- Les données partagées

Ce programme étant implémenté sur une machine multiprocesseur à mémoire partagée , nous pouvons utiliser des données partagées , donc communes à toutes les tâches . Ces données appartiennent à des commons fortran . Lorsque ces données sont constantes , c'est-à-dire qu'elles ne sont jamais modifiées , elles peuvent être consultées par plusieurs tâches simultanément . En revanche , les données qui sont modifiées ne peuvent être consultées ou modifiées que dans le cadre de sections critiques , contrôlées par des sémaphores .

Les données du problème à résoudre (matrices A , b et c) sont partagées et ne sont consultées qu'en lecture . Nous ne leur associons donc pas de sémaphores .

Les données relatives à l'arborescence évoluent au cours de l'exécution du programme . Il est donc impératif de les protéger . Les nœuds actifs sont codés dans un tableau et ordonnés par un arbre de recherche . Chaque nœud est caractérisé par la donnée de :

- le nombre de variable actives dans le problème associé
- le nombre de contraintes actives
- la liste des variables fixées à 1 ou à 0
- la liste des contraintes éliminées
- les coefficients du multiplicateur composite optimal associé au nœud père du problème
- l'évaluation du nœud
- les pointeurs le liant aux autres nœuds de l'arbre de recherche

Nous devons stocker toutes ces données dans le tableau , pour chacun des nœuds actifs . En fait nous prenons un tableau bidimensionnel par donnée . Par exemple , le tableau X1 , à deux dimension , contient la liste des variables fixées à 1 ou à 0 des nœuds actifs .

BIBLIOGRAPHIE

[CC] CCVR , "Vectorisation"
Rapport technique , Palaiseau , 1983 .

[CR] CRAY RESEARCH INC. , "Multitasking user's guide"
CRAY computer systems technical notes , SN-022 , 1984 .

[ERH] ERHEL J. , "CREM user's manual"
Rapport technique , n° 25 , INRIA , 1983 .

[FA1] FAYARD D. , PLATEAU G.
"Contribution à la résolution des programmes mathématiques en nombres entiers"
Thèse d'Etat-Université des Sciences et Techniques de Lille,1979.

[FA2] FAYARD D. , PLATEAU G.
"An algorithm for the solution of the 0-1 knapsack problem"
Computing 28 , 1982 , p. 269-287 .

[FI] FISHER M.L.
"The Lagrangian relaxation method of solving the multiconstraint 0-1 knapsack to optimality"
Management Science 27 , 1981 , p. 1-18.

[FR1] FREVILLE A.
"Heuristiques et réduction pour les problèmes mathématiques en variables 0-1 à contraintes d'inégalité"
Thèse de 3ème cycle , Université des Sciences et Techniques de Lille , 1983 .

[FR2] FREVILLE A. , PLATEAU G.
"Heuristics and reduction methods for multiple constraints 0-1 linear programming problems"
EJOR 24 , 1986 , p. 206-215 .

[FR3] FREVILLE A. , PLATEAU G.
"Hard 0-1 multiknapsack test problems for size reduction methods"
Rapport de recherche 72 , Université de Paris-Nord , 1987 .

[GA] GAVISH B. , PIRKUL H.
"Efficient algorithms for solving the multiconstraints 0-1 knapsack problems to optimality"
Mathematical Programming 31 , 1985 , p. 78-105 .

[GE1] GEOFFRION A.M.
"Lagrangian relaxation for integer programming"
Mathematical Programming Study 2 , 1974 , p. 82-114 .

[GL1] GLOVER F.
"Surrogate constraints duality in mathematical programming"
Operations Research 23 , 1975 , P. 879-919 .

[HOC] HOCKNEY R.W. , JESSHOPE C.R.
"Parallel Computers"
Adam Hilger , Ltd Bristol , 1981 .

[LAU] LAURENT P. , VERLEYE C.
"Procédures arborescentes sur machines parallèles et sur réseaux de processeurs"
Mémoire d'ingénieur de l'IIE , 1985 .

[PET] PETERSEN C.C.
"Computational experience sith variants of the Balas algorithm applied to the selection of R and D projects"
Management Science 13 , 1967 , p. 736-750 .

[PR1] PLATEAU G. , ROUCAIROL C.
"A parallel algorithm for the size reduction of the 0-1 multiknapsack problem"
A paraître , 1988 .

[PR2] PLATEAU G. , ROUCAIROL C. , VALABREGUE I.
"Algorithm PR^2 for the parallel size reduction of the 0-1 multiknapsack problem"
Rapport de recherche INRIA , 1988 .

[ROU] ROUCAIROL C.
"Du parallèle au séquentiel : La recherche arborescente et son application à la programmation quadratique en variables 0 1"
Thèse d'Etat , Université Pierre et Marie Curie - Paris VI , 1987.

[SA] SATIN S. , KARWAN M.H. , RARDIN R.L.
"A new surrogate dual search procedure in integer programming"
Naval Research Logistic 34 , 1984 , p. 431-450 .

[SA1] SATIN S. , KARWAN M.H. , RARDIN R.L.
"Surrogate duality in a branch-and-bound procedure for integer programming"
EJOR 33 , 1988 , p. 326-333 .

[SEN] SENJU S. , TOYODA Y.
"An approach to linear programming with 0-1 variables"
Management Science 15 , 1968 , p. 196-207 .

[SH] SHIH W.
"A branch-and-bound method for the multiconstraint 0-1 multiknapsack problem"
EJOR 30 , 1979 .

[VA] VALABREGUE I.
"Multiknapsack en parallèle - Phase de réduction"
Mémoire d'ingénieur de l'IIE , 1987 .

[WEI] WEIGARTNER H.M. , NESS D.N.
"Methode for the solution of the multidimensional 0-1 knapsack problem"
Operations Research 15 , 1967 , p. 83-103 .

[YAO] YAO A.C.
"On parallel computation for the knapsack problem"
JACM 29 , 1982 , p. 898-903 .

ANNEXE LE PROGRAMME PR²88

Cette annexe comprend une documentation relative au programme (spécification des variables et des procédures) .

Programme parallèle pour la résolution du multiknapsack

I- La phase de réduction

1- Variables partagées et locales

L'appellation variable partagée qui apparaît dans cette annexe désigne une variable commune à plusieurs tâches parallèles , tandis qu'une variable locale désigne une variable réservée strictement à la tâche qui la déclare .

Afin de les différencier des variables locales , les variables partagées sont notées en majuscules .

2- Définition des variables partagées

Les données du problème

NP	nombre de processeurs de la machine
NV	nombre de variables du problème du départ
NC	nombre de contraintes du problème du départ
C(1..NV)	tableau des coefficients de la fonction économique
A(1..NC,1..NC)	matrice des coefficients du système des contraintes
B(1..NC)	vecteur associé au second membre du système des contraintes
MIN	valeur d'une solution réalisable connue

Les variables caractérisant un état du problème

NVA	nombre de variables non encore fixées
NVA1	nombre de variables actives à la fin de la phase 1
NCA	nombre de contraintes non éliminées
NCA1	nombre de contraintes actives à la fin de la phase 1
NTA	nombre de tâches actives
NT	nombre de tâches lancées en parallèle, par la tâche initiale
NTA1	nombre total de tâches venues au rendez-vous
AR(1..NCA,1..NVA) , BR(1..NCA) , CR(1..NVA)	contiennent les données du problème réduit , correspondent à A , B , C
F0 , F1	signifie qu'au moins une variable a été fixée à 0 ou à 1 au cours de l'itération
MODIF	une variable a été fixée au cours de l'itération
MOPT	le minorant correspond à la solution optimale du problème

X1 , X2 ensemble des variables ,qui peuvent se trouver dans trois états (fixées à 1, fixées à 0 , non fixées)
X1 est modifié au fur et à mesure des fixations , X2 mémorise l'état du vecteur à l'itération précédente

VPLUS somme des coûts des variables fixées à 1

PLUS(1..NTA) somme des coûts des variables fixées à 1 par chaque tâche

TX(1..NP,1..NV) tableau des résultats de chaque tâche
tx(i,j) = 2 : la varialbe j n'a pas été fixée
tx(i,j) = 0 ou 1 , la variable j a été fixée précvédemment à 1 ou à 0
tx(i,j) = 3 , la tâche i a fixé la variable j à 0 au cours de kla phase courante
tx(i,j) = 4 , la tâche i a fixé la variable j à 1

DTX(1..NTA,1..NV) synthèse des résultats enregistrés dans TX(1..NTA,1..NV)

TC(1..NC) indique l'état des contraintes (1 : contrainte libre , 2 contrainte élue par une tâche , 0 contrainte éliminée)

CODTC(1..NC) liste des contraintes éliminées lors de la phase de fixation des variables (par le test R2)

TCR(1..NCA+INDICE) indique l'état des knapsacks outils en phase 1 (0 non traité , 1 traité)

CPT nombre de tâches arrivées au rendez-vous

NBFINI nombre de tâches arrivées en fin de phase 2

FIN(1..NP) état des tâches : active ou inactive

MINOPT(1..NP) détection de la terminaison générale par une tâche (vrai si une tâche a détecté une contradiction)

TABW(1..40,1..NCA) tableau des multiplicateurs composites engendrés par l'algorithme du sous-gradient

INDICE nombre de multiplicateurs composites engendrés par l'algorithme du sous-gradient

I_EVT numéro de l'évènement bloquant (I_EVT = 0 lors d'une itération d'ordre impair , et 1 lors d'une itération d'ordre pair)

EVT(0..1) tableau d'évènements envoyés alternativement lorsque les processus actifs ont terminé une itération de la phase 1

CPTEM , FINSEM , TCSEM sémaphores d'exclusion mutuelle sur les variables respectivement CPT , NBFINI , TC , TCR et NCA

MAJ(1..NTA) tableau utilisé pour la mise en commun des résultats au cours d'une itération de la phase 1

SEMAMAJ sémaphore d'accès à la variable MAJ

BORNE nombre minimum de variables qu'il faut qu'une tâche ait fixé pour qu'elle communique ses résultats aux autres tâches

3- Les sections critiques

Les variables , TCR consultée et modifiée par les tâches parallèles en phase 1 , TC et NCA , consultées et modifiées par les tâches parallèles en phase 2 , mémorisent l'état de contraintes ; on dispose du sémaphore TCSEM pour en verouiller l'accès .

Les compteurs des tâches arrivées en fin de phase 1 (CPT) et en fin de phase 2 (NBFINI) , les variables X1 et MAJ au cours de la phase 1 sont les seules autres variables exigeant un traitement en exclusion mutuelle .

4- Définition des variables locales

no_ta	numéro de la tâche
iter	numéro du prochain rendez-vous = numéro de l'itération
succes1	succès du test R1
succes2	succès du test R2
no_evt	numéro de l'évènement attendu par les tâches qui sont arrivées au rendez-vous avant le dernière tâche
nelim	nombre de variables fixées par une tâche et par un knapsack outil
nelim1	nombre de variables fixées par la tâche depuis la dernière mise en commun des résultats

5- Définition des procédures

AGNES(données : AR , BR , CR , NVA , NCA ; résultats : X1)
détermine une solution réalisable du problème en appliquant l'heuristique agnes 2

CHELIM(données : y1 , r1 ; résultats : xx1)
Lors du test des relations binaires appliqué à un knapsack outil engendré par une contrainte , applique le tests V1 aux matrices y1 et r1 (coûts réduits et valeurs de relaxations) et rend la liste des variables fixées .

CHELIM(données : y1 , r1 ; résultats : xx1)
Lors du test des relations binaires appliqué à un knapsack outil engendré par un multiplicateur composite , applique le tests V1 aux matrices y1 et r1 (coûts réduits et valeurs de relaxations) et rend la liste des variables fixées .

COND(données : TC , TCSEM , i , j ; résultats : j) : entier
alloue un knapsack outil j à une tâche pour l'élimination de la contrainte i

CONTROL(données : TX ; résultats : MODIF , MOPT , F0 , F1)
met en commun les résultats de toutes les tâches (compare les listes de variables fixées) à la fin de chaque itération (phase de fixation des variables)

ECRES écriture des résultats de la phase de réduction dans un fichier

ELIMK(données : k , i , AR , BR , CR) : logique
applique les tests d'élimination de la contrainte i au knapsack outil engendré par la k^{iem} contrainte . Rend vrai si la contrainte a été éliminée .

ELIMWK(données : k , i , AR , BR , CR) : logique
applique les tests d'élimination de la contrainte i au knapsack outil engendré par le k^{iem} multiplicateur composite . Rend vrai si la contrainte a été éliminée .

ESSAICT(données : no_v , valeur , nk , ligne , no_ta ; résultat y1 , r1)
simule la fixation de la variable x_{no_v} à valeur lors du test des relations binaires appliqué au knapsack outil engendré par une contrainte .

ESSAICTW(données : no_v , valeur , nk , ligne , no_ta ; résultat y1 , r1)
simule la fixation de la variable x_{no_v} à valeur lors du test des relations binaires appliqué au knapsack outil engendré par un mulitplicateur composite .

FIXVV(données : no-ta , AR , BR , CR , MIN , VPLUS , i ; résultats :minopt , TX , nelim)
applique les tests de fixation des variables au knapsack outil engendré par la contrainte i

FIXW(données : no-ta , AR , BR , CR , MIN , VPLUS , i ; résultats :minopt , TX , nelim)
applique les tests de fixation des variables au knapsack outil engendré par le knapsack outil généré par le i^{iem} multiplicateur composite.

LECDON procédure de lecture des données du problème (A , B , C, MIN , NT)

MAJ(données : TX , X1 ; résultats : DTX , TX , X1)
enregistre les résultats réunis de toutes les tâches actives en phase 1 dans le tableau DTX et met à jour le tableau X1 en fonction des résultats produits par les tâches , puis recopie X1 dans TX.

MAJR1(données : DTX ; données-résultats : NVA , NCA , AR , BR , CR)
met à jour les données du problème réduit par une itération de la phase 1

MAJR2 (données : NV , NC , X1 , TC , A , B , C ; résultats : AR , BR , CR)
met à jour AR et BR à la fin de la phase 2 et met à jour les données du problème réduit à la fin de la phase 1

MOTAB(données : xx1 ; résultats : y1 , r1)
Lors du test des relations binaires appliqué à un knapsack outil engendré par une contrainte , met à jour les matrices des coûts réduits (y1) et des valeurs des relaxations (r1) en fonction des variables fixées (xx1)

MOTABW(données : xx1 ; résultats : y1 , r1)
Lors du test des relations binaires appliqué à un knapsack outil engendré par un multiplicateur composite , met à jour les matrices des coûts réduits (y1) et des valeurs des relaxations (r1) en fonction des variables fixées (xx1)

NOUVEAU(données : DTX , X1 , X2 , TCR ; résultats : TCR)
met à jour TCR pour l'itération suivante , en particulier , détecte les contraintes actives qui n'ont pas été affectées par l'élimination des variables .

NUM-C(données : ir , NC , TC) : entier
calcule l'indice de la contrainte ir du problème réduit , par rapport au problème d'origine .

NUM-R(données : is , TC) : entier
calcule l'indice de la contrainte is du problème d'origine , par rapport au problème réduit

NUM-V(données : ir , NV , X1) : entier
calcule l'indice de la variable ir dans le problème réduit , par rapport au problème d'origine .

NUM-V0(données : ik , no-c , no-ta , TX , X1) : entier
calcule l'indice de la variable ik du problème de knapsack traité par la tâche no-ta , dans le sous-programme FIXVAR , par rapport aux données du problème source .

R1(données : AR , BR , ; résultats AR , BR , CR , X1 , NVA , succès1)
effectue le test trivial R1

R2(données : AR , BR ; résultats : TCR , TC , NCA , succès2)
effectue le test R2

R3(données : AR , X1 ; résultats : NVA ,; X1 , AR , CR)
effectue le test R3

REDULAG1(données : AR , BR , CR , NVA , NCA ; résultats : TABW)
génère une sous-suite de multiplicateurs composites en applique-ant un algorithme du type du sous-gradient au problème réduit .

RESOL10(données AR , BR , CR ; résultats X1)
résoud le problème réduit explicitement lorsqu'il reste moins de 10 variables actives et rend la solution du problème d'origine .

SOLVE1(données : a , b , c :résultats : ib , valeur)
résoud un knapsack en continu et rend sa variable de base et son évaluation
a , b et c sont les matrices des données du knapsack (entiers)

SOLVE2(données : a , b , c :résultats : ib , valeur)
résoud un knapsack en continu à coefficients réels et rend sa variable de base et son évaluation
a , b et c sont les matrices des données du knapsack (réels)

TROUVE1(données TCR , TCSEM ; résultats : i)
alloue un knapsack outil libre à une tâche lors de la phase d'élimination des variables

TROUVE2(données : TC , TCSEM ; résultats : i)
alloue une contrainte à une tâche lors de la phase de l'élimination des contraintes

II- La phase de résolution du problème

1 Définition des variables partagées

Les données du problème

NCA	nombre de contraintes du problème réduit
NVA	nombre de variables du problème réduit
NTA	nombre de tâches actives
AR , BR , CR	matrices des coefficients du problème
MIN	minorant de la valeur du problème réduit
PLACE	nombre de lignes des tableaux représentant l'arbre

Les variables représentant l'arborescence des noeuds actifs

TNVC(1..PLACE) tableau contenant le nombre de variables actives pour chaque nœud
TNCC(1..PLACE) tableau contenant le nombre de contraintes actives pour chaque nœud
TX1(1..PLACE,1..NVA) tableau contenant , pour chaque nœud , la liste des variables fixées
TCC(1..PLACE,1..NCA) tableau contenant , pour chaque nœud , la liste des contraintes éliminées
TW(1..PLACE,1..NCA)tableau contenant , pour chaque nœud , les coefficients du multiplicateur composite associé au père du nœud
TVB(1..PLACE) tableau contenant l'indice de la variable de base de chacun des nœuds
TEVAL(1..PLACE) tableau contenant l'évaluation de chacun des nœuds
TNS(1..PLACE) tableau contenant , pour chaque nœud , le numéro de la ligne du nœud suivant
TNP(1..PLACE) tableau contenant , pour chaque nœud , le numéro de la ligne du nœud précédent

TFEUILLE , QNOEUDS , TNOEUDS : pointeurs permettant la gestion de l'arbre
BI borne inférieure

Sémaphores et évènements

SEMAARBRE , SEMAPLACE , SEMAATPL , SEMAATPB , SEMAFIN , SEMASYNC : sémaphores
I_EVTF , I_EVTS(1..NTA) , I_EVTP : numéros de l'évènement courant
EVTF(0..1) , EVTP(0..1) , EVTS(1..NY_TA,0..1) : tableaux d'évènements

Variables statistiques

NBFAITS(1..2,1..NTA) nombre de nœuds traités par chaque tâche
NBMAX nombre maximum de nœuds actifs présents dans l'arborescence
ATMAX(1..NTA) nombre de fois que chaque tâche a été mise en attente (pour le contrôle de la répartition du travail)
NAS(1..NTA) nombre de nœuds traités par chaque tâche avant d'avoir atteint la solution optimale
NTOTAL(1..NTA) nombre de nœuds parcourus par chaque tâche

2- Variables locales à chaque tâche

arn(1..NCA,1..NVA) , brn(1..NCA) , crn(1..NVA)	matrices associées au problème du nœud traité par la tâche
nvc , ncc	nombre de variables et de contraintes du problème traité par la tâche
aw	matrice des coefficients du knapsack associé au problème (relaxation composite) ($B_k(w)$)
bw	second membre de la contrainte du knapsack ($B_k(w)$)
lx1(1..NVA)	liste des variables fixées à 1 et à 0 (pour la problème (B_k))
tcc(1..NVA)	liste des contraintes éliminées (pour la problème (B_k))
leval	évaluation du nœud
lambda	tableau des multiplicateurs Lagrangiens
y	tableau des coûts réduits
r	tableau des valeurs des relaxations

3- Définition des procédures

DEMPLACE ; entier
alloue une ligne libre du tableau représentant l'arborescence à une tâche

ECRIRE liste les nœuds de l'arborescence

ECRNOEUD ecrit dans un fichier les caractéristiques d'un nœud

ELIMINER(données : ligne , arbre ; résultats arbre)
supprime le nœud se trouvant sur la ligne ligne de l'arborescence

METTREFILE(données ligne , arbre ; résultat : arbre)
lie le nœud ligne aux autres nœuds de l'arborescence

RANGER(données caractéristiques d'un nœud , arborescence ; résultats : arborescence)
Insère un nouveau nœud dans l'arborescence

RESOLBB(données AR , BR , CR , NVA , NCA , MIN ; résultats : X1)
procédure de résolution du problème du multiknapsack . Rend la solution optimale du problème réduit .

SUPPRIMER(données valeur , arbre ; résultats arbre)
Elimine les nœuds de l'arborescence ayant une évaluation inférieure à valeur

RESOLPAR(données : numéro de tâche)
Traitement effectué par chaque tâche : en fait
- demander des nœuds libres
- traiter le nœud
- ranger les nœuds créés

REDULAG(données arn , brn , crn , nvc , ncc , w ; résultats w)
génère le multiplicateur composite (sous-gradient) associé au problème traité

Choix effectués pour le codage en fortran 77

1 Pour une version unique

Le compilateur Fortran 77 , tel qu'il existe actuellement sur le CRAY 2 , permet de déclarer des variables communes aux différentes tâches lancées en parallèle dans un même programme - instruction COMMON . En revanche , il n'est pas possible de déclarer de variables communes à des sous-programmes appelés par des tâches différentes , sans que celles-ci ne soient nécéssairement partagées par ces tâches . L'instruction "Task Common" , qui existe à cet effet n'est pas en service .

Le logiciel CREM qui a permis de mettre au point le programme permet de déclarer : des variables partagées entre les tâches - instruction COMMON$ - , et des variables locales à une tâche , partagées par des sous-programmes -instruction COMMON .

Nous nous sommes efforcés de développer une version unique du programme , dans les limites des contraintes logicielles , telles que celle évoquée ci-dessus .

2- Limiter le nombre de paramètres des sous-programmes

Le passage de paramètres est plus rapide par l'instruction COMMON . Pour les variables partagées il est possible de les déclarer dans un COMMON , dans le sous-programme appelé et appelant . Mais les variables locales à une tâche doivent être appelées comme paramètres du sous-programme appelé . Une autre solution réside dans la déclaration d'un COMMON de tableaux de variables , chaque occurence du tableau correspondant aux données d'une tâche .

3- Optimisation du code

Pour accélérer l'exécution du code FORTRAN , nous nous sommes efforcés de désemboîter les boucles imbriquées , de placer les boucles les plus longues au niveau le plus bas , et de faire appel à des fonctions optimisées de la bibliothèque $SCILIB .

Parmi les fonctions optimisées de cette bibliothèque , nous avons surtout utilisé les suivantes :

ISEARCH(n,vecteur,inc,objet) fonction de type entier donnant l'indice du premier élément du vecteur ayant pour valeur objet .

SAXPY(N,SA,SX,INCX,SY,INCY) calcule Y = AX + Y

Chaque fois que le nombre d'itérations d'une boucle est suffisant , il devient intéressant de la vectoriser .

La séquence : si (var.op.expression) var = expression empêche la vectorisation d'une boucle . Elle peut être optimisée par une instruction de la forme var = function(var , expression) , où function est une version des fonctions MAX/MIN . Ainsi , on ramène une boucle non vectorisable à une boucle vectorisable .

Pour les instructions de type si(expression logique) var = expression , nous avons employé la fonction **CVGMT** :

var = CVGMT(expression ,var,condition)

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique